Knowl Inf Syst manuscript No.
(will be inserted by the editor)

A Safe Region based Approach to Moving *K*NN Queries in Obstructed Space

Chuanwen Li · Yu Gu · Jianzhong Qi · Rui Zhang · Ge Yu

Received: 26 Apr 2013 / Revised: 15 Oct 2014 / Accepted: 24 Oct 2014 /

Abstract The moving k nearest neighbor (MkNN) query has been studied extensively. Most of the studies assume no obstacle in the space. However, obstacles like rivers, buildings and private properties commonly exist in the space and one may need to go around the obstacles to reach his/her nearest neighbors. In this paper, we study the moving kNN query in obstructed space with no predefined query object trajectory. We take a safe region based approach to solve this problem. In particular, we propose a method to compute a safe region w.r.t. a data object. In this safe region, the query object can move freely while the data object is kept in the query object's kNN set. By combining the safe regions of the data objects near the query object, we formulate an overall safe region where the query object's kNN set keeps stable. We propose an algorithm based on the safe regions to process the moving kNN query in obstructed space. Extensive experiments show that the proposed algorithm significantly reduces the communication and the computation costs for query processing. Our algorithm outperforms a baseline algorithm by up to two orders of magnitude under various settings.

Keywords Moving nearest neighbor query, obstructed space, safe region, fixed-rank region

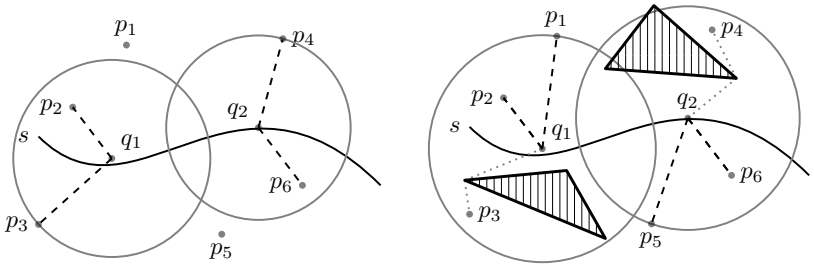
1 Introduction

In recent years, there has been growing needs for Location Based Services (LBS) and an important application is finding nearest objects such as gas stations, ATMs, restaurants and so on in mobile scenarios [10, 13, 15]. For example, a tourist driving a car wants to find a list of nearest gas stations using a GPS-equipped mobile phone. The query is sent to a server and needs to be answered continuously while the position of the query object (i.e., the mobile phone) changes. The server maintains a database of various spatial objects, which is used to process the query and the results are sent back to the mobile phone. This is a typical example of the *Moving k Nearest Neighbor query (M*k*NN)* [22, 23, 35]. Given a set of data points

Chuanwen Li · Yu Gu (✉) · Ge Yu
The School of Information Science & Engineering, Northeastern University, Shenyang, Liaoning, China.
E-mail: {lichuanwen, guyu, yuge}@ise.neu.edu.cn

Jianzhong Qi · Rui Zhang
The Department of Computing and Information Systems, University of Melbourne, Victoria, 3052, Australia.
E-mail: {jianzhong.qi, rui.zhang}@unimelb.edu.au

P (gas stations, restaurants, etc.) and a moving query point q (a car, a mobile phone, etc.), an MkNN query retrieves the k nearest data points of q from P for every timestamp – this means that the answer set may need to be updated as q moves. Figure 1(a) gives an example where $k = 2$, and the curve s is the trajectory of q. When q moves to position q_1, the 2NN set of q is $\{p_2, p_3\}$, and at q_2, the 2NN set of q becomes $\{p_4, p_6\}$.



(a) MkNN (b) Obstructed MkNN

Fig. 1: Example of MkNN and obstructed MkNN

There have been various studies on the MkNN query as well as its variants, such as the moving visible NN query [11] and the MkNN query on predefined linear trajectories [9]. In this paper we study an important variant of the MkNN query, the *M*k*NN query in obstructed space*. For simplicity we call it the *obstructed M*k*NN query*. This query considers the obstacles (such as rivers and buildings) in the space that the query point may need to go around, which may result in different kNN answers. Figure 1(b) gives an example. It has the same data points and query trajectory as those of Figure 1(a). In addition it has two obstacles shown as the shaded polygons. At location q_1, the 2NN set of q is now $\{p_1, p_2\}$ instead of $\{p_2, p_3\}$. This is because now p_3 is farther than p_1 from q due to the distance added by the detour around the obstacles. For the same reason, at location q_2, the 2NN set of q is $\{p_5, p_6\}$ instead of $\{p_4, p_6\}$.

A recent study [9] showed how to process the obstructed MkNN query assuming that the query point moves along a predefined linear trajectory. This assumption is quite restrictive. In practice, it is more common that the query point does not have a predefined trajectory, which brings in a significant challenge to query processing. Specifically, in this case we cannot predict the kNN set of the query point based on its predefined trajectory. Instead, we have to keep monitoring the movement of the query point and recomputing the kNN set repeatedly, which may lead to high communication and computation costs.

In this paper, we address this challenge and propose an efficient algorithm to process the *obstructed M*k*NN query with no predefined query trajectory*.

Our basic idea is inspired by V*-Diagram [22], the state-of-the-art method for the MkNN query with no predefined query trajectory in unobstruced space (i.e., a space without obstacles). V*-Diagram is a safe region based technique. Its safe region is formed by two types of regions, the *safe region w.r.t. a data point* and the *fixed-rank region*. We define similar regions in obstructed space to address the obstructed MkNN query. However, our work is not a trivial extension of V*-Diagram [22] because our work requires exploiting geometric properties entailed by obstacles, which bring in significant difficulties in defining the safe regions. In obstructed space, the distance between two objects has to take into account the

extra distance for bypassing the obstacles. Hence, the safe regions usually have irregular shapes. We address these challenges and make the following contributions:

- This is the first work that addresses the obstructed M*k*NN query with no predefined query trajectory. We formulate two types of regions, namely the *obstructed safe region w.r.t. a data point* and the *obstructed fixed-rank region*, to form a safe region called the *obstructed integrated safe region* for processing the query.
- We exploit the properties of the obstructed integrated safe region to reduce the query processing cost, and obtain a highly efficient algorithm to process the obstructed M*k*NN query.
- We perform a comprehensive experimental study on the proposed algorithm. The results show that our algorithm outperforms the baseline algorithm by up to two orders of magnitude.

This paper is an extended version of our earlier paper [16]. There we proposed a safe region based approach to process the obstructed M*k*NN query. In this paper we extend our work by (i) significantly improving the performance of our obstructed M*k*NN query processing algorithm through formalizing the *obstructed disk* and the *essential auxiliary set*, which are used in the *obstructed known region* to reduce the number of obstacle vertices to be checked in safe region computation (details are in Section 4.1 and Section 7); (ii) conducting experiments on the improved algorithm and comparing it with a baseline algorithm adapted from a related study [9]; (iii) handling more types of obstacles; (iv) providing detailed theoretical foundations to the proposed approach, including formal definitions and proofs to the key techniques and complexity analysis; (v) improving the overall presentation of the paper, including more explanatory figures and clarifications.

The remainder of this paper is organized as follows: We review related studies in Section 2. In Section 3, we provide preliminaries of the study. We formulate the obstructed safe region w.r.t a data point and the obstructed fixed-rank region in Sections 4 and 5, respectively. We present the obstructed integrated safe region and an algorithm to process the obstructed M*k*NN query in Section 6. We discuss how to use the essential auxiliary set to constrain the search space in Section 7. In Section 8 we extend the proposed algorithm to different types of obstacles. We conduct an experimental study in Section 9 and conclude the paper in Section 10.

2 Related Work

We first review studies on spatial queries over moving objects in general. Then we review M*k*NN queries in unobstructed space and obstructed space, respectively.

2.1 Spatial Queries over Moving Objects

Spatial queries over moving objects have been studied extensively. For example, Šaltenis et al. [28] propose the Time Parameterized R-tree (TPR-tree), which indexes moving points as linear functions of time. Tao et al. [26] use the TPR-tree to process the time-parameterized queries, which query moving objects satisfying certain time-parameterized predicates. Hu et al. [13] propose a safe region based framework to monitor spatial queries over moving objects for client-sever based systems. In such a system, each moving object is a client. It knows the current query result and does not need to report its location updates to the server

until the updates may cause query result change. Mokbel et al. [19, 20] also propose frameworks to monitor spatial queries over moving objects. Their frameworks process the queries incrementally by monitoring the effect of each object location update on the query answer. Benetics et al. [2] study *reverse nearest neighbor (RNN) queries* over moving objects. Xia et al. [32] propose a *six-region* approach for the query type. More recently, Cheema et al. [3] propose a safe region based approach to process the *reverse k nearest neighbor (RkNN) query* over moving objects. Li et al [17] study moving *k*NN query in weighted regions. Eunus Ali et al. [8] study moving range queries and Zhang et al. [37] study predictive range queries. Zhang et al. [38] propose the intersection join query over moving objects and Ward et al. [30] propose a GPU based algorithm to process the query. This query finds the intersecting pairs from two sets of moving objects. These studies have different problem settings from ours and their solutions are inapplicable.

2.2 M*k*NN Queries in Unobstructed Space

As a major type of spatial queries, the *k*NN query has attracted a large body of studies (e.g., [14]). In the past decade, *k*NN queries over moving objects, i.e, the M*k*NN queries, have become more popular. Most of the existing studies on the M*k*NN queries consider unobstructed space [13, 15, 22]. For example, Tao et al. [27] study finding the NNs for each point on a line segment, which can be seen as finding the NNs of a point object moving on the line segment. Song et al. [25] use a sampling based approach to reduce the M*k*NN query processing cost in the expense of query result accuracy.

Many studies use safe region based approaches to process the query. The k^{th}-order Voronoi Diagram (*k*VD) [23] is an example. This method requires expensive precomputation and updates for the safe regions. The *Retrieve-Influence-Set* *k*NN algorithm (RIS-*k*NN) [35] alleviates the high precomputation cost of *k*VD by computing a k^{th}-order Voronoi cell locally, but it still has high update cost. Specifically, it requires issuing a number of (six on average) expensive time-parameterized *k*NN queries every time the query point exits the current safe region. Other studies (e.g., [27]) assume a predefined linear trajectory of the query point. In this case, the safe region is reduced to a line segment on the predefined trajectory, and can be determined by the bisectors between the query point and its nearby data points with low costs. Nutanong et al. [22] propose the V*-Diagram technique, which maintains some extra nearest neighbors to formulate a safe region. As the extra nearest neighbors maintained serve as a "cache", this technique does not need to retrieve new data points every time the query point exits the current safe region. It requires much less frequent data retrieval as well as much cheaper data access cost per data retrieval, and it handles dynamically changing k values. As V*-Diagram is closely related to our proposed technique, we will describe it in detail in Section 3.3.

Besides the safe region based approaches, Li et al. [18] uses *influential neighbor set* to bound *k*NN sets, which reduces *k*NN recalculation cost while retaining the lowest *k*NN recalculation frequency. Yu et al. [34] use grid indexes for M*k*NN queries, which produce exact query answers but with a delay in the time. Xiong et at. [33] also use grid indices. They propose an incremental computation approach to maintain an *answer region* for each query, which updates according to the object location updates. The answer region is then used to derive the query answer. Mouratidis et al. [21] propose a threshold based approach for M*k*NN queries. This approach tries to reduce the communication overhead between the query processor and the moving objects. Hsueh et al. [12] use *Location Information Ta-*

bles and obtain a partition based lazy update algorithm to reduce the object location update processing costs.

Since these studies do not consider obstacles in the M*k*NN query, their methods do not apply to our problem.

2.3 M*k*NN Queries in Obstructed Space

In computational geometry, shortest path problems in obstructed space are extensively studied [24]. The shortest path between two points in an obstructed space can be computed using any conventional shortest path algorithm based on the *visibility graph* (detailed in Section 3.1).

Zhang et al. propose the *obstructed NN (ONN)* query [36] that finds the static nearest neighbors of a static query point in obstructed space. Their proposed method is the state-of-the-art solution to static obstructed *k*NN queries. It can be executed repeatedly as a sampling based approach to approximate a continuous answer to the obstructed M*k*NN query. A more detailed discussion on the obstructed NN query on static objects can be found in [31].

Gao and Zheng [9] consider the M*k*NN queries in obstructed space. They assume that the query point is moving on a predefined line segment. Gao et al. also study the reverse *k* nearest neighbor query in the obstructed space with predefined query trajectory [10]. They assume static data objects and a query object q moving along a given line segment, and propose pruning heuristics to reduce the query processing costs. These heuristics are based either on the distance between the *static* data objects and the line segment where q moves along, or on the position relationship between the *static* data objects, an obstacle and q. They do not apply because we assume no predefined query trajectory, which is more practical in real applications.

Wang et al. [29] study the continuous visible *k*NN queries where both the query object and the data objects are moving. This work is less relevant and will not be discussed further.

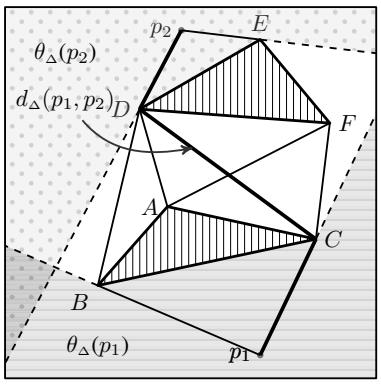


Fig. 2: Visibility graph

3 Preliminaries

We assume a set of obstacles O (convex polygons with non-zero extents), a set of static data points P, and a moving query point q. All these objects are in an Euclidean space. The

Table 1. Frequently Used Symbols

Symbol	Meaning
k	The number of queried data points.
O	The set of obstacles
V	The set of all vertices of the obstacles in O
P	The set of data points
q	The moving query point
q_b	The last location where the kNN set is updated
x	The number of auxiliary data points.
z	The $(k+x)^{th}$ nearest data point of q.
$\theta_\Delta(p)$	The visible region of a point p.
$\beta(p_1,p_2)$	The unobstructed dominant region of p_1 over p_2.
$\beta_\Delta(p_1,p_2)$	The obstructed dominant region of p_1 over p_2.
$d(p,q)$	The unobstructed distance between p and q.
$d_\Delta(p,q)$	The obstructed distance between p and q.
$\phi(p,l)$	The unobstructed disk with p as the center and l as the radius.
$\phi_\Delta(p,l)$	The obstructed disk with p as the center and l as the radius.
$W(q_b,z)$	The unobstructed known region centered at q_b corresponding to z.
$W_\Delta(q_b,z)$	The obstructed known region centered at q_b corresponding to z.
$\varepsilon(p_1,p_2,l)$	The ellipse with p_1 and p_2 as its two foci and l as its major axis length.
$\varepsilon_\theta(p_1,p_2,l)$	The visible ellipse part.
$\omega(q_b,p,l)$	The unobstructed safe region w.r.t. a data point p.
$\omega_\Delta(q_b,p,l)$	The obstructed safe region w.r.t. a data point p.
$\eta(L)$	The unobstructed fixed-rank region w.r.t. a data point list L.
$\eta_\Delta(L)$	The obstructed fixed-rank region w.r.t. a data point list L.
$\Omega(q_b,L)$	The unobstructed integrated safe region w.r.t. q_b and L.
$\Omega_\Delta(q_b,L)$	The obstructed integrated safe region w.r.t. q_b and L.

obstructed MkNN query returns the kNN set of q continuously (i.e., at every timestamp), where k is a given query parameter.

The distance between q and a data point p is defined as the length of the shortest path from q to p without crossing any obstacle, denoted by $d_\Delta(q,p)$. The shortest path may be a series of connected line segments. For example, in Figure 2, there are two obstacles ABC and DEF. The shortest path between p_1 and p_2 consists of three line segments $\overline{p_1C}$, $\overline{CD}$ and $\overline{Dp_2}$. Thus, $d_\Delta(p_1,p_2) = |\overline{p_1C}| + |\overline{CD}| + |\overline{Dp_2}|$, where $|\cdot|$ denotes the line segment length.

Next, we describe three basic concepts to help compute $d_\Delta(q,p)$ and process the obstructed MkNN query, namely, the *visibility graph*, the *bisector of two points* and the V^*-*Diagram*. We summarize the frequently used symbols in Table 1, where "$_\Delta$" is used to indicate that obstacles are involved in the definition of a symbol.

3.1 Visibility Graph

The visibility graph is based on the *visible region* [23], which is the region visible to a point p, denoted by $\theta_{\Delta}(p)$. Here "visible" means not being blocked by any obstacle. Formally,

$$\theta_{\Delta}(p) = \{t | \overline{pt} \cap [o \backslash \partial o] = \emptyset, \forall o \in O\}, \tag{1}$$

where $\overline{pt}$ denotes the line segment connecting two points p and t, and $o\backslash\partial o$ denotes an obstacle o excluding its boundary. This exclusion is because p and t can still be visible to each other if $\overline{pt}$ intersects the boundary of o but does not cross the o.

The graph containing the visible regions of multiple data points is called a *visibility graph*. Figure 2 shows an example, where the dotted area and the horizontal lined area denote the visible regions of p_1 and p_2, respectively. Based on this visibility graph, we can identify the shortest path between p_1 and p_2 (i.e., $\overline{p_1 C D p_2}$). This shortest path is computed by constructing a graph on the vertices of the obstacles, the source point and the destination point, and then applying the Dijkstra's algorithm. The detailed shortest path computation process can be found in [23].

3.2 Bisector of Two Points

The *bisector* of two points p_i and p_j, denoted by $b(p_i, p_j)$, is defined as a line or curve on which a point t has the same distance to both p_i and p_j. Formally,

$$b(p_i, p_j) = \{t \,|\, d(t, p_i) = d(t, p_j)\} \quad i \neq j. \tag{2}$$

When obstacles are involved, the definition becomes:

$$b_{\Delta}(p_i, p_j) = \{t \,|\, d_{\Delta}(t, p_i) = d_{\Delta}(t, p_j)\} \quad i \neq j. \tag{3}$$

We illustrate $b_{\Delta}(p_i, p_j)$ with Figure 3, where a bisector $b_{\Delta}(p_1, p_2)$ is formed by a series of curve segments $\langle \widehat{b_1 b_2}, \widehat{b_2 b_3}, \widehat{b_3 b_4}, \widehat{b_5 b_6}, \widehat{b_6 b_7}, \widehat{b_7 b_8} \rangle$ (the bold curve). This is different from the bisectors in unobstructed space where they are usually straight lines, and the difference is because of the "detoured" distance caused by the obstacles.

The bisector of p_i and p_j divides the space into two regions. In one region, every point has a smaller distance to p_i than to p_j. This region is called the *dominant region* [23] of p_i over p_j, denoted by $\beta(p_i, p_j)$.

$$\beta(p_i, p_j) = \{t \,|\, d(t, p_i) < d(t, p_j)\} \quad i \neq j \tag{4}$$

Similarly, in the other region, every point has a smaller distance to p_j than to p_i, and the region is called the dominant region of p_j over p_i, denoted by $\beta(p_j, p_i)$.

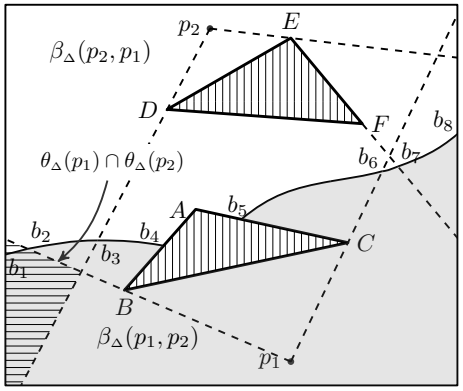


Fig. 3: The bisector of p_1 and p_2

In obstructed space the definition becomes:

$$\beta_\Delta(p_i,p_j)=\{t\,|\,d_\Delta(t,p_i)<d_\Delta(t,p_j)\}\quad i\neq j \tag{5}$$

In Figure 3, the gray region represents $\beta_\Delta(p_1,p_2)$ and the white region represents $\beta_\Delta(p_2,p_1)$.

3.3 V*-Diagram

V*-Diagram [22] is the state-of-the-art solution for MkNN queries. By fully exploiting the knowledge of the query point location and the safe region, V*-Diagram significantly reduces the number of times the query point exits the safe region (in other words, the number of times the query is re-evaluated by the server) and the cost of re-evaluating the query (processing a kNN query).

The safe region used in the V*-Diagram is called the *Integrated Safe Region* (ISR), which is the intersection of two types of regions: (i) the *safe region w.r.t. the k^{th} nearest data point*, and (ii) the *fixed-rank region* of the $k+x$ nearest data points, where x is the number of auxiliary data points maintained in the query processor to reduce safe region recomputation.

The V*-Diagram MkNN algorithm computes the integrated safe region as follows. It first finds the query point's $(k+x)$ nearest data points. Let q_b denote the current position of q and z denote its $(k+x)^{th}$ nearest data point. Then a *known region* is computed as a disk centered at q_b with the radius being $d(q_b,z)$, where $d(q_b,z)$ denotes the unobstructed distance between q_b and z.

Step 1. The *safe region w.r.t. a data point p*, denoted by $\omega(q_b,p,d(q_b,z))$, is then computed as a region that, when the query point q moves inside the region, p is nearer to q than any data point p' outside the known region. Formally,

$$\omega(q_b,p,d(q_b,z))=\{q'|d(q',p)\leq d(q',p')\} \tag{6}$$

Since $d(q_b,q')+d(q',p')\geq d(q_b,p')$ (triangle inequality), the definition of $\omega(q_b,p,d(q_b,z))$ can be tightened by by replacing $d(q',p')$ with $d(q_b,p')-d(q_b,q')$:

$$\omega(q_b,p,d(q_b,z))=\{q'|d(q',p)\leq d(q_b,p')-d(q_b,q')\}. \tag{7}$$

Point p' is outside the know region, i.e., $d(q_b,p')\geq d(q_b,z)$. Thus, the equation is further simplified to be:

$$\begin{aligned}\omega(q_b,p,d(q_b,z)) &= \{q'|d(q',p) \leq d(q_b,z) - d(q_b,q')\} \\ &= \{q'|d(q',p) + d(q_b,q') \leq d(q_b,z)\}.\end{aligned} \tag{8}$$

This equation shows that the safe region w.r.t. p is effectively an ellipse in the Euclidean space where q_b and p are its two foci and $d(q_b,z)$ is its major axis length (MAL). We denote this ellipse by $\varepsilon(q_b,p,d(q_b,z))$.

As long as q is inside the intersection of the safe regions w.r.t. the $k+x$ data points in the known region, i.e., $\bigcap_{i=1}^{k+x}\omega(q_b,p_i,d(q_b,z))$, we guarantee that any data point p' outside the known region cannot be closer to q than any of those $k+x$ data points.

Step 2. The V*-Diagram algorithm further computes a region called the *fixed-rank region* where the order of distances of the $k+x$ data points to q does not change, either. Formally, the fixed-rank region of a list L_{k+x} of $k+x$ ranked data points, $\eta\ \langle p_1,p_2,...,p_{k+x}\rangle$, is defined as the intersection of the dominant region of p_i over p_{i+1} ($i \in [1..k+x-1]$):

$$\eta\langle p_1,p_2,...,p_{k+x}\rangle = \bigcap_{i=1}^{k+x-1}\beta(p_i,p_{i+1}) \tag{9}$$

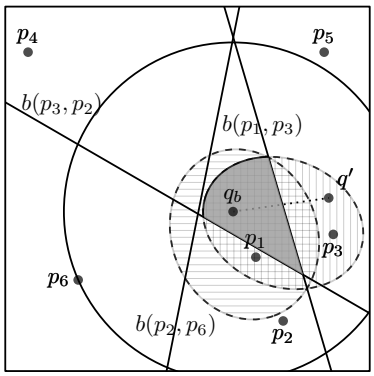


Fig. 4: Integrated safe region ($k=2$, $x=2$)

Step 3. The intersection of the safe regions w.r.t. the $k+x$ data points and the fixed-rank region is the *Integrated Safe Region (ISR)*, denoted by $\Omega(q_b,L_{k+x})$. Formally, $\Omega(q_b,L_{k+x}) = \eta(L_{k+x}) \cap (\bigcap_{i=1}^{k+x}\omega(q_b,p_i,d(q_b,z)))$, where p_k denotes the k^{th} nearest data point of q. This computation can be simplified to the following equation [22]:

$$\Omega(q_b,L_{k+x}) = \eta(L_{k+x}) \cap \omega(q_b,p_k,d(q_b,z)). \tag{10}$$

Figure 4 shows an example where $k=2$ and $x=2$. When the query point q is at the initial location q_b, a 4NN search retrieves the 4 nearest data points $\langle p_1,p_3,p_2,p_6\rangle$. The ellipses filled with horizontal lines and vertical lines denote $\omega(q_b,p_1,d(q_b,p_6))$ and $\omega(q_b,p_3,d(q_b,p_6))$, respectively. Then as long as q remains in the grey region $\eta\langle p_1,p_3,p_2,p_6\rangle$ $\cap\omega(q_b,p_3,d(q_b,p_6))$, the 2NN of q will not change.

3.4 Solution Framework

In V*-Diagram, the ISR is computed in unobstructed space. It can be expressed as a set of ellipses and lines and hence can be easily maintained. When the space is obstructed, substantial difficulties are added to the formulation of such regions. It is unclear what the safe region is like and how it can be computed and maintained.

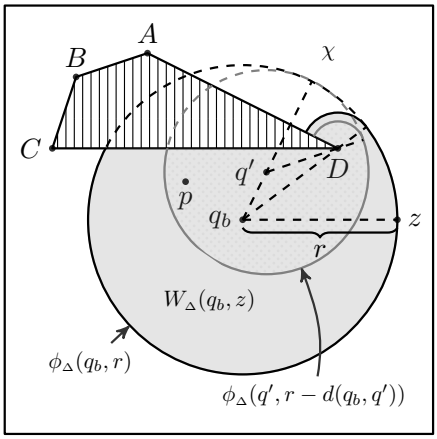


Fig. 5: Obstructed known region

In the following sections, we will address these difficulties and present obstructed versions of the regions used in V*-Diagram, namely, *the obstructed known region, the obstructed safe region w.r.t a data point, the obstructed fixed-rank region and the obstructed integrated safe region*. Based on these regions, we propose the *Obstructed M*k*NN algorithm* to process the obstructed M*k*NN query as follows.

- Initialization. When an obstructed M*k*NN query is issued, the algorithm first computes a list L_{k+x} of $k+x$ nearest data points of q, where the x extra data point serve as a "cache" to reduce the number of safe region recomputation.
- Maintenance. At every timestamp, the new position of q arrives. The algorithm uses the obstructed safe region and the obstructed fixed-rank region to determine whether a safe region recomputation is required and which data points need to be accessed for the recomputation. Specifically, if q is still in the obstructed safe regions w.r.t the data points in L_{k+x} but not the fixed-rank region, then the recomputation can be done on only the points in L_{k+x}. Otherwise a kNN query is issued to update L_{k+x} and get the new kNN set of q. The above process repeats and the obstructed M*k*NN set of q is generated continuously.

4 Obstructed Safe Region w.r.t. a Data Point (OSRD)

4.1 Obstructed Known Region

When the query point q is at position q_b and its $(k+x)^{th}$ nearest data point is z, the obstructed known region of q, denoted by $W_\Delta(q_b,z)$, is defined as a region where the obstructed distance from any point t to q_b is less than or equal to the obstructed distance $d_\Delta(q_b,z)$. Figure 5 gives an example, where the shaded polygon *ABCD* denotes an obstacle and the grey region denotes an obstructed known region.

As can be seen from the figure, the obstructed known region is not a disk in Euclidean space. Next we describe how to compute the obstructed known region through a concept called the "obstructed disk".

We use $\phi(q_b,r)$ to denote a disk with center q_b and radius r in unobstructed space. Similarly, we use $\phi_\Delta(q_b,r)$ to denote an *obstructed disk* in obstructed space where the points' distances to q_b are less than or equal to r. Figure 6 shows an example, where the two shaded triangles denote obstacles and the grey region denotes an obstructed disk.

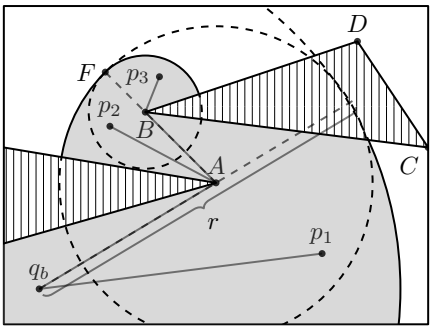


Fig. 6: Obstructed disk

We have the following observations on an obstructed disk $\phi_\Delta(q_b,r)$.

1. In the visible region $\theta_\Delta(q_b)$ (i.e., the area not being blocked by any obstacle), $\phi_\Delta(q_b,r)$ and $\phi(q_b,r)$ cover the same area.
2. For an obstacle vertex v whose obstructed distance to q_b, $d_\Delta(v,q_b)$, is less than r, the obstructed disk $\phi_\Delta(v,r-d_\Delta(v,q_b))$ is fully covered by $\phi_\Delta(q_b,r)$. This is because, for any point t in $\phi_\Delta(v,r-d_\Delta(v,q_b))$, its obstructed distance to q_b is at most $r-d_\Delta(v,q_b)+d_\Delta(v,q_b)=r$.
   In Figure 6, the grey region enclosed by the dotted circle centered at vertex A denotes obstructed disk $\phi_\Delta(A,r-d_\Delta(A,q_b))$. It is fully covered by $\phi_\Delta(q_b,r)$.
3. Let $\langle q_b,v_1,...,v_i,t\rangle$ be the vertices on the shortest path from q_b to t, where $v_1,...,v_i$ are some vertices of the obstacles in the space. Then t must be in the visible region $\theta_\Delta(v_i)$, where v_i is the last obstacle vertex that the shortest path passes before reaching t. If t is in $\phi_\Delta(q_b,r)$, then t must be in $\phi_\Delta(v_i,r-d_\Delta(v_i,q_b))$ and hence $\phi(v_i,r-d_\Delta(v_i,q_b))$. This is because $d_\Delta(v_i,t)=d_\Delta(q_b,t)-dis(q_b,v_i)\leq r-d_\Delta(v_i,q_b)$.

In Figure 6, for point p_3, vertex B is the last vertex on its shortest path from q_b. Therefore, it is in $\theta_\Delta(B)$ and $\phi_\Delta(B, r - d_\Delta(q_b, B))$ and hence in $\phi(B, r - d_\Delta(q_b, B))$, which is denoted by the dotted circle centered at B.

Based on these observations, we can construct an obstructed disk $\phi_\Delta(q_b, r)$ by combining the obstructed disks of the obstacle vertices enclosed by $\phi(q_b, r)$ (i.e., a circular region centered at q_b with a radius of r), as formalized by Theorem 1.

Theorem 1 *Let $V_{\phi(q_b,r)}$ be the set of obstacle vertices enclosed by the unobstructed disk $\phi(q_b, r)$. We have:*

$$\phi_\Delta(q_b, r) = \bigcup_{v \in \{q_b\} \cup V_{\phi(q_b,r)}} \phi(v, r - d_\Delta(q_b, v)) \cap \theta_\Delta(v). \tag{11}$$

Proof (i) "$\phi_\Delta(q_b, r) \Rightarrow \bigcup_{v \in \{q_b\} \cup V_{\phi(q_b,r)}} \phi(v, r - d_\Delta(q_b, v)) \cap \theta_\Delta(v)$". If a point t is in $\phi_\Delta(q_b, r)$, then the last vertex v on the shortest path from q_b to t must be in $\{q_b\} \cup V_{\phi(q_b,r)}$. Based on Observation 3, we have t in $\phi(v, r - d_\Delta(q_b, v)) \cap \theta_\Delta(v)$.

(ii) "$\phi_\Delta(q_b, r) \Leftarrow \bigcup_{v \in \{q_b\} \cup V_{\phi(q_b,r)}} \phi(v, r - d_\Delta(q_b, v)) \cap \theta_\Delta(v)$". If a point t is in region $\phi(v, r - d_\Delta(p, v)) \cap \theta_\Delta(v)$, where $v \in \{p\} \cup V_{\phi(q_b,r)}$, then by Observation 2 we know it must be inside $\phi_\Delta(q_b, r)$.

Given Equation 11, we can now compute the known region in an obstructed space, $W_\Delta(q_b, z)$, as $\phi_\Delta(q_b, d_\Delta(q_b, z))$ (cf. Figure 5).

4.2 Definition of OSRD

Based on the definition of the obstructed known region, we can define the *Obstructed Safe Region w.r.t. a Data point p (OSRD)*. OSRD is a region where the movement of q will not cause p to be removed from the $k+x$ nearest neighbors of q. As shown in Figure 5, if q has moved from q_b to q', for p to stay as one of the $k+x$ NNs of q, the distance between p and q' must be less than or equal to the distance between q' and any point χ outside the obstructed known region, i.e., $d_\Delta(q', p) \leq d_\Delta(q', \chi)$. Since the obstructed known region is defined by a "radius" of $d_\Delta(q_b, z)$, we have $d_\Delta(q', \chi) + d_\Delta(q', q_b) \geq d_\Delta(q_b, z) \Rightarrow d_\Delta(q', \chi) \geq d_\Delta(q_b, z) - d_\Delta(q', q_b)$. Therefore, we need $d_\Delta(q', p) \leq d_\Delta(q_b, z) - d_\Delta(q', q_b)$ to guarantee that p stays as one of the $k+x$ NNs of q.

Formally, OSRD is defined as follows.

Definition 1 (Obstruct safe region w.r.t. a Data point)
Given a data point p and the $(k+x)^{th}$ nearest data point of q_b, denoted by z, the obstructed safe region w.r.t. p, denoted by $\omega_\Delta(q_b, p, d_\Delta(q_b, z))$, is defined as:

$$\begin{aligned} \omega_\Delta(q_b, p, d_\Delta(q_b, z)) &= \{q' | d_\Delta(q', p) \leq d_\Delta(q_b, z) - d_\Delta(q_b, q')\} \\ &= \{q' | d_\Delta(q', p) + d_\Delta(q_b, q') \leq d_\Delta(q_b, z)\}. \end{aligned} \tag{12}$$

We can see that Equation (12) is very similar to Equation (8), which defines an elliptical shaped safe region in obstructed space. We call it an obstructed ellipse. Even though $\omega_\Delta(q_b, p, d_\Delta(q_b, z))$ is not a traditional ellipse, we still call q_b and p its two foci, and $d_\Delta(q_b, z)$ its major axis length (MAL). We replace $d_\Delta(q_b, z)$ by l in the following discussion for ease of presentation.

4.3 Computation of OSRD

The definition of OSRD specified a point set but does not provide a clear way to represent the boundary of the set in closed forms such as an ellipse. In this subsection, we propose a method to compute OSRD based on the following three lemmas.

The first lemma establishes that in the region visible to both q_b and p, OSRD covers the same region with or without considering the obstacles.

Lemma 1 *In the intersection of the visible regions of q_b and p, the obstructed safe region $\omega_\Delta(q_b,p,l)$ and the safe region $\omega(q_b,p,l)$ computed by omitting the obstacles cover the same area, i.e., $\omega_\Delta(q_b,p,l)\cap\theta_\Delta(q_b)\cap\theta_\Delta(p)=\omega(q_b,p,l)\cap\theta_\Delta(q_b)\cap\theta_\Delta(p)$.*

Proof In $\theta_\Delta(q_b)\cap\theta_\Delta(p)$, there is no obstacles. The shortest distance from any point to q_b or p in the obstructed space is the same as that without the obstacles. Therefore, the obstructed safe region in $\theta_\Delta(q_b)\cap\theta_\Delta(p)$ is the same as its unobstructed counterpart.

Recall that $\omega(q_b,p,l)$ is an ellipse denoted by $\varepsilon(q_b,p,l)$. Thus, $\omega(q_b,p,l)\cap\theta_\Delta(q_b)\cap\theta_\Delta(p)$ is the part of $\varepsilon(q_b,p,l)$ that is visible from both q_b and p. We call it the *visible ellipse part* (VEP) and denote it by $\varepsilon_\theta(q_b,p,l)$.

The second lemma establishes that the OSRD of p fully covers the OSRD of certain points in the OSRD of p.

Lemma 2 *Given two different points q' and p' in $\omega_\Delta(q_b,p,l)$, if $d_\Delta(q_b,q')+d_\Delta(p,p')+d_\Delta(q',p')\le l$, then $\omega_\Delta(q',p',l-d_\Delta(q_b,q')-d_\Delta(p,p'))\subseteq\omega_\Delta(q_b,p,l)$.*

Proof If $d_\Delta(q_b,q')+d_\Delta(p,p')+d_\Delta(q',p')\le l$, then $l-d_\Delta(q_b,q')+d_\Delta(p,p')\ge d_\Delta(q',p')>0$. Therefore, it guarantees that $\omega_\Delta(q',p',l-d_\Delta(q_b,q')-d_\Delta(p,p'))$ is an OSRD.

Next we prove that a point t in $\omega_\Delta(q',p',l-d_\Delta(q_b,q')-d_\Delta(p,p'))$ must also be in $\omega_\Delta(q_b,p,l)$. By definition we have $d_\Delta(t,q')+d_\Delta(t,p')\le l-d_\Delta(q_b,q')-d_\Delta(p,p')$. Thus,

$$l\ge d_\Delta(t,q')+d_\Delta(q_b,q')+d_\Delta(t,p)+d_\Delta(p,p')\ge d_\Delta(t,q_b)+d_\Delta(t,p). \tag{13}$$

Therefore, t is in $\omega_\Delta(q_b,p,l)$.

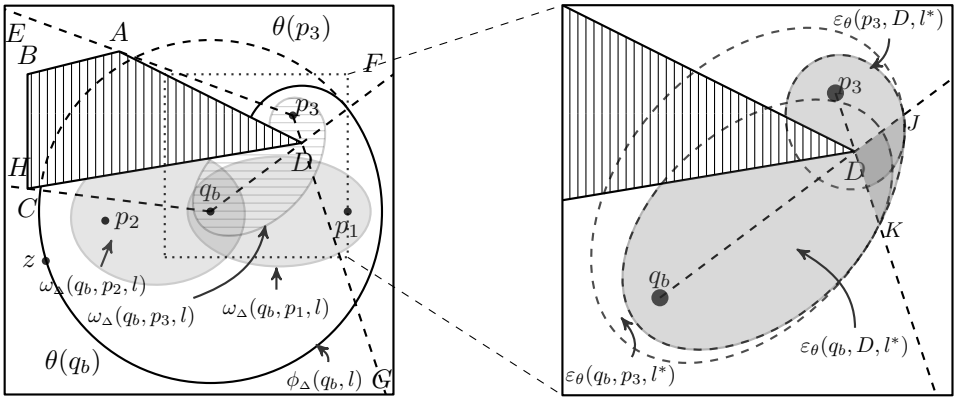
Based on the two lemmas above, we have that $\omega_\Delta(q_b,p,l)$ can be computed by the intersection of the VEPs of all sub-OSRDs satisfying Lemma 2. Since the points satisfying Lemma 2 are infinite, we need to identify a subset of such points whose OSRDs together fully cover and hence form $\omega_\Delta(q_b,p,l)$. The following lemma gives us such a subset.

Lemma 3 *Let $V_{\omega(q_b,p,l)}$ be a set containing all obstacle vertices in $\omega(q_b,p,l)$. Then for any point t in $\omega_\Delta(q_b,p,l)$, there exist two points v_1 and v_2 in $\{q_b,p\}\cup V_{\omega(q_b,p,l)}$ such that $t\in\theta_\Delta(v_1)\cap\theta_\Delta(v_2)$.*

Proof If t is visible to q_b, we let q_b be v_1 and have $t\in\theta_\Delta(v_1)$. Otherwise the shortest path from q_b to t must pass certain obstacle vertices. Let v_1 be the last obstacle vertex that the shortest path passes. Then $t\in\theta_\Delta(v_1)$. Meanwhile, v_1 is closer to q_b than t. Thus, v_1 is in $\omega(q_b,p,l)$ and hence in $V_{\omega(q_b,p,l)}$.

Similarly, we can identify v_2 on the shortest path from p to t. Thus, we have two points $v_1,v_2\in\{q_b,p\}\cup V_{\omega(q_b,p,l)}, t\in\theta_\Delta(v_1)\cap\theta_\Delta(v_2)$.

Lemma 3 gives every point t in $\omega_\Delta(q_b,p,l)$ points from $\{q_b,p\}\cup V_{\omega(q_b,p,l)}$ to form ellipses to enclose t. There is at least one of such ellipses whose VEP is part of $\omega_\Delta(q_b,p,l)$. We use Lemma 2 to filter out those points whose ellipses' VEPs are not part of $\omega_\Delta(q_b,p,l)$, and the rest of the points are used to compute $\omega_\Delta(q_b,p,l)$ as the following theorem suggests.



(a) Simple safe region (b) $\omega_{\Delta}(q_b, p_3, l)$

Fig. 7: Obstructed safe regions

Theorem 2 *Let v_1, v_2 be two points from $\{q_b, p\} \cup V_{\omega(q_b,p,l)}$. If $d_{\Delta}(v_1, q_b) + d_{\Delta}(v_2, p) + d_{\Delta}(v_1, v_2) \leq l$, then $\varepsilon(v_1, v_2, l - d_{\Delta}(v_1, q_b) - d_{\Delta}(v_2, p)) \cap \theta_{\Delta}(v_1) \cap \theta_{\Delta}(v_2)$ is part of $\omega_{\Delta}(q_b, p, l)$. Formally,*

$$\omega_{\Delta}(q_b, p, l) = \bigcup_{v_1, v_2 \in \{q_b, p\} \cup V_{\omega(q_b,p,l)}} \varepsilon(v_1, v_2, \hat{l}(v_1, v_2)) \cap \theta_{\Delta}(v_1) \cap \theta_{\Delta}(v_2), \tag{14}$$

where $d_{\Delta}(v_1, q_b) + d_{\Delta}(v_2, p) + d_{\Delta}(v_1, v_2) \leq l$ and $\hat{l}(v_1, v_2) = l - d_{\Delta}(v_1, q_b) - d_{\Delta}(v_2, p)$.

Proof The correctness of the theorem is guaranteed by the lemmas above straightforwardly.

Theorem 2 suggests that $\omega_{\Delta}(q_b, p, l)$ consists of several VEPs. The foci of these VEPs can be q_b, p or the obstacle vertices enclosed by $\omega(q_b, p, l)$, i.e., the safe region computed without considering the obstacles.

The combination of the VEPs can have three different cases.

- **Case 1:** There is no obstacle in $\omega_{\Delta}(q_b, p, l)$ and thus, $\omega_{\Delta}(q_b, p, l)$ and $\omega(q_b, p, l)$ cover exactly the same area. (cf. Figure 7(a), $\omega_{\Delta}(q_b, p_1, l)$).
- **Case 2:** There are obstacles intersecting $\omega_{\Delta}(q_b, p, l)$'s corresponding safe region computed without considering the obstacles, $\omega(q_b, p, l)$, but there is no obstacle vertex in $\omega(q_b, p, l)$. In this case, $\omega_{\Delta}(q_b, p, l) = \omega(q_b, p, l) \cap \theta_{\Delta}(q_b) \cap \theta_{\Delta}(p)$ (cf. Figure 7(a), $\omega_{\Delta}(q_b, p_2, l)$).
- **Case 3:** There are obstacle vertices in $\omega(q_b, p, l)$. Then we need Equation 14 to compute $\omega_{\Delta}(q_b, p, l)$ as a combination of VEPs. In Figure 7(a), OSRD $\omega_{\Delta}(q_b, p_3, l)$ illustrates this case. We show the detailed computation of $\omega_{\Delta}(q_b, p_3, l)$ in Figure 7(b), where we first compute three VEPs $\varepsilon_{\theta}(q_b, p_3, \hat{l}(q_b, p_3))$, $\varepsilon_{\theta}(q_b, D, \hat{l}(q_b, D))$, $\varepsilon_{\theta}(p_3, D, \hat{l}(p_3, D))$ and then compute their union to form $\omega_{\Delta}(q_b, p_3, l)$.

We further simplify the computation of $\omega_{\Delta}(q_b, p, l)$ for the regions that are visible to more than two vertices that satisfy the conditions in Theorem 2. As the following corollary suggests, in the overlapping visible region of three points v_1, v_2 and v_3 satisfying the conditions in Theorem 2, if the obstructed shortest path of v_2 and v_3 passes v_1 (i.e., the

shortest path of v_2 and v_3 is $\overline{v_2...v_1...v_3}$), then $\varepsilon_\theta(v_2,v_3,\hat{l}(v_2,v_3))$ covers $\varepsilon_\theta(v_1,v_2,\hat{l}(v_1,v_2))$ and $\varepsilon_\theta(v_1,v_3,\hat{l}(v_1,v_3))$. An example is in Figure 7(b). Among q_b, p_3 and D, the obstructed shortest path between q_b and p_3 passes D. Thus, in the overlapping visible region JDK, ellipse $\varepsilon(q_b,p_3,\hat{l}(q_b,p_3))$ fully covers $\varepsilon(q_b,D,\hat{l}(q_b,D))$ and $\varepsilon(p_3,D,\hat{l}(p_3,D))$. Therefore, we can avoid computing parts of $\omega_\Delta(q_b,p,l)$ covered by more than one VEPs based on the following corollary .

Corollary 1 *Given three points v_1, v_2 and v_3 in $\{q_b,p\}\cup V_\omega(q_b,p,l)$, where v_2 lies on the shortest path from v_1 to q_b and v_3 lies on the shortest path from v_1 to p. We have:*

$$\begin{cases} \varepsilon(v_2,v_3,\hat{l}(v_2,v_3))\cap\varepsilon(v_1,v_3,\hat{l}(v_1,v_3)) = \varepsilon(v_1,v_3,\hat{l}(v_1,v_3)) \\ \varepsilon(v_2,v_3,\hat{l}(v_2,v_3))\cap\varepsilon(v_1,v_2,\hat{l}(v_1,v_2)) = \varepsilon(v_1,v_2,\hat{l}(v_1,v_2)), \end{cases} \tag{15}$$

which means $\varepsilon(v_2,v_3,\hat{l}(v_2,v_3))$ contains $\varepsilon(v_1,v_2,\hat{l}(v_1,v_2))$ and $\varepsilon(v_1,v_3,\hat{l}(v_1,v_3))$.

Proof Let t be a point in $\varepsilon(v_1,v_3,\hat{l}(v_1,v_3))$. Since $d(t,v_2)\le d(t,v_1)+d_\Delta(v_1,v_2)$ and $d_\Delta(v_1,q_b)=d_\Delta(v_1,v_2)+d_\Delta(v_2,q_b)$, we have $d(t,v_2)+d_\Delta(v_2,q_b)+d(t,v_3)+d_\Delta(v_3,q_b)\le d(t,v_1)+d_\Delta(v_1,q_b)+d(t,v_3)+d_\Delta(v_3,q_b)\le l$, which ensures that t is in $\varepsilon(v_2,v_3,\hat{l}(v_2,v_3))$.

Similarly we can prove that a point in $\varepsilon(v_1,v_2,\hat{l}(v_1,v_2))$ is in $\varepsilon(v_2,v_3,\hat{l}(v_2,v_3))$.

4.4 Algorithm for Computing OSRD

In this subsection, we present our OSRD computing algorithm *OSRDC* based on Theorem 2, as shown in Algorithm 1. The algorithm starts with computing the visibility graph and identifying the obstructed known region (lines 1 to 3). Then all vertices in the obstructed known region plus q_b and p are added to a list V_{all}, which is to be used to compute the VEPs to form $\omega_\Delta(q_b,p,l)$ (line 4). The points in V_{all} are sorted by their obstructed distances to q_b (line 6) to reduce the computation based on Corollary 1. Then the algorithm uses a two-layer for-loop to evaluate every pair of points in V_{all} that can contribute to $\omega_\Delta(q_b,p,l)$ (lines 7 to 12) based on Theorem 2. The VEPs that form $\omega_\Delta(q_b,p,l)$ are computed and added to $\omega_\Delta(q_b,p,l)$ during this process. When the for-loop ends, $\omega_\Delta(q_b,p,l)$ has been computed and is returned by the algorithm.

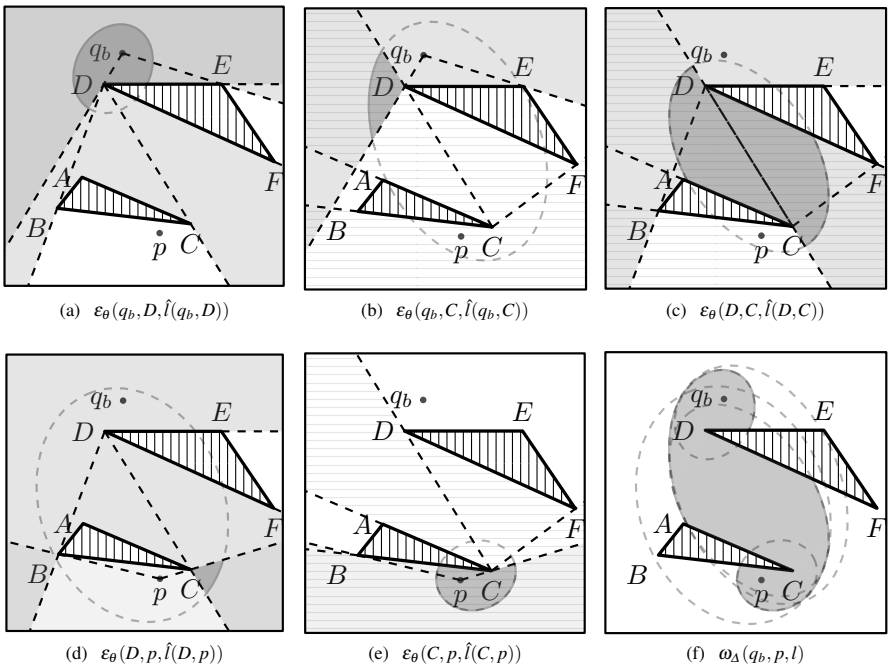
An example. Figure 8 shows how an OSRD is computed through computing the VEPs, where the grey region and the horizontal-lined region in each subfigure denote the visible regions of the two points selected for VEP computation. The points selected for VEP computation are sorted by their distances to q_b. The first point selected by the outer layer for-loop is q_b, since it has 0 distance to q_b. Then the first point selected by the inner for-loop is D, which is the closest to q_b. The VEP w.r.t. q_b and D, $\varepsilon_\theta(q_b,D,\hat{l}(q_b,D))$, is computed and shown by the grey ellipse part in Figure 8(a). Point C is the next point selected by the inner for-loop and $\varepsilon_\theta(q_b,C,\hat{l}(q_b,C))$ is computed and shown in Figure 8(b). Then the inner for-loop ends and the outer for-loop gets to point D (and the inner for-loop restarts at C). The above process repeats and VEPs $\varepsilon_\theta(D,C,\hat{l}(D,C))$, $\varepsilon_\theta(D,p,\hat{l}(D,p))$ and $\varepsilon_\theta(C,p,\hat{l}(C,p))$ are computed as shown in Figure 8(c), Figure 8 (d) and Figure 8(e), respectively. The combination of all VEPs computed gives us $\omega_\Delta(q_b,p,l)$ (cf. Figure 8(f)).

Complexity. In the algorithm, we first add the vertices in $\phi_\Delta(q_b,l)$ to V_{all}. Identifying the obstacle vertices in $\phi_\Delta(q_b,l)$ with a spatial index (e.g., R*-tree) takes $O(\log n)$ time on average [5], where n denotes the number of all obstacle vertices. We denote the number of vertices in V_{all} by $|V_{all}|$, which is determined by the obstacle density, the data point density, and the query parameter k. Then the two-layer for-loop has a time complexity of $O(|V_{all}|^2)$. In all, the algorithm takes $O(\log n+|V_{all}|^2)$ time.

Algorithm 1: Computing OSRD

Input : Obstacles O, data point p, latest query position q_b
Output: $\omega_\Delta(q_b,p,l)$

Compute the visibility graph based on O
Compute the $(k+x)^{th}$ nearest data point z
$l \leftarrow d_\Delta(q_b,z)$
$V_{all} \leftarrow V_{\phi_\Delta(q_b,l)} \cup \{q_b,p\}$
$\omega_\Delta(q_b,p,l) \leftarrow \emptyset$
Sort the vertices in V_{all} by their obstructed distances to q_b in an ascending order
foreach v_i *in* V_{all} *except the last one* **do**
    foreach v_j *in* V_{all} *after* v_i **do**
        if $d_\Delta(v_i,q_b)+d_\Delta(v_j,p)+d_\Delta(v_i,v_j) \le l$ **then**
            $\hat{l}(v_i,v_j) = l - d_\Delta(v_i,q_b) - d_\Delta(v_j,p)$
            $\varepsilon_\theta \leftarrow \varepsilon(v_i,v_j,\hat{l}(v_i,v_j)) \cap \theta_\Delta(v_i) \cap \theta_\Delta(v_j)$
            $\omega_\Delta(q_b,p,l) \leftarrow \omega_\Delta(q_b,p,l) \cup \varepsilon_\theta$
return $\omega_\Delta(q_b,p,l)$



(a) $\varepsilon_\theta(q_b,D,\hat{l}(q_b,D))$ (b) $\varepsilon_\theta(q_b,C,\hat{l}(q_b,C))$ (c) $\varepsilon_\theta(D,C,\hat{l}(D,C))$

(d) $\varepsilon_\theta(D,p,\hat{l}(D,p))$ (e) $\varepsilon_\theta(C,p,\hat{l}(C,p))$ (f) $\omega_\Delta(q_b,p,l)$

Fig. 8: Computation of OSRD

5 Obstructed Fixed-Rank Region

In this section we present the *obstructed fixed-rank region (OFR)* to guarantee that the order of distances of the $k+x$ data points in the obstructed known region to q does not change.

Definition 2 (Obstructed fixed-rank region) The obstructed fixed-rank region of a list L_{k+x} of $k+x$ data points ordered ascendingly by their distances to the query point at q_b,

denoted by $\eta_\Delta\langle p_1, p_2, ..., p_{k+x}\rangle$, is a region in an obstructed space that, when q moves inside the region, the distance rank of the data points in L_{k+x} to q is fixed. It is computed as the intersection of the obstructed dominant region of p_i over p_{i+1} ($i \in [1..k+x-1]$):

$$\eta_\Delta\langle p_1, p_2, ..., p_{k+x}\rangle = \bigcap_{i=1}^{k+x-1} \beta_\Delta(p_i, p_{i+1}) \tag{16}$$

Figure 9 shows an example. There are three data points p_1, p_2 and p_3 and two obstacles ABC and DEF. The three bisectors $b_\Delta(p_1,p_2)$, $b_\Delta(p_2,p_3)$ and $b_\Delta(p_1,p_3)$ divide the data space into six OFRs. Each OFR corresponds to a distance rank of the three data points. For instance, for any point in $\eta_{\Delta 1} = \eta_\Delta\langle p_3, p_1, p_2\rangle = \beta_\Delta(p_3,p_1) \cap \beta_\Delta(p_1,p_2) \cap \beta_\Delta(p_3,p_2)$, p_3 is the nearest data point, while p_2 is the second and p_1 is the third.

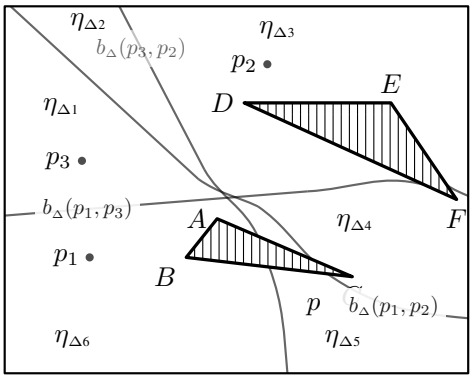


Fig. 9: OFR

If we compute the OFR for the $k+x$ data points in the obstructed known region, then unless q moves out of this OFR, we do not need to recompute the kNN set. However, computing and storing an exact OFR is too costly due to the curve bisectors caused by obstacles (cf. Figure 9). Therefore, instead of computing and storing the OFR, we check whether the dominating relationship of the $k+x$ data points, i.e., $d_\Delta(q,p_i) \le d_\Delta(q,p_{i+1})$, holds for every pair of data points $\langle p_i, p_{i+1}\rangle$ in L_{k+x}. If it does, then q is still in the current OFR; otherwise, q has moved out of the OFR and re-ordering for the $k+x$ data points is needed.

Dominating relationship check. Next we present an algorithm to check whether $d_\Delta(q,p_i) \le d_\Delta(q,p_{i+1})$ holds. We call it the *dominating relationship check (DRC)* algorithm, as summarized in Algorithm 2. The algorithm has two stages: (i) initialization, and (ii) evaluation.

The *initialization stage* (lines 1 to 3) computes a visibility graph on p_i and p_{i+1}. First, it computes the obstructed distance $d_\Delta(p_i,p_{i+1})$. Then the obstructed disks $\phi_\Delta(p_i, d_\Delta(p_i,p_{i+1}))$ and $\phi_\Delta(p_{i+1}, d_\Delta(p_i,p_{i+1}))$ are computed. The visibility graph is computed based on the obstacles whose vertices are in both disks because these are the only obstacles whose vertices can contribute to $d_\Delta(q,p_i)$ or $d_\Delta(q,p_{i+1})$. This way the visibility graph computation cost is constrained.

The *evaluation stage* (lines 4 to 6) uses Dijkstra's algorithm [7] to compute shortest paths from q to p_i and p_{i+1} and hence $d_\Delta(q,p_i)$ and $d_\Delta(q,p_{i+1})$. Whether $d_\Delta(q,p_i)$ is less than or equal to $d_\Delta(q,p_{i+1})$ is returned.

Complexity. The initialization stage can be computed once at the start and then the visibility graph can be reused in the following dominating relationship check. Therefore, the cost is amortized. For line 1, computing $\phi_\Delta(p_i, d_\Delta(p_i, p_{i+1}))$ and $\phi_\Delta(p_{i+1}, d_\Delta(p_i, p_{i+1}))$ requires $O(n_V \log n_V)$ time, where n_V denotes the number of all obstacle vertices. Identifying all vertices in $\phi_\Delta(p_i, d_\Delta(p_i, p_{i+1}))$ and $\phi_\Delta(p_i, d_\Delta(p_i, p_{i+1}))$ by an R*-tree takes $O(\log n_V)$ time. Let the number of these vertices be n_{V1}. Then constructing a visibility graph with them takes $O(n_{V1}^2 \log n_{V1})$ time [6]. In all, the initialization stage takes $O(n_V \log n_V + \log n_V + n_{V1}^2 \log n_{V1})$ time.

The evaluation stage involves two shortest path computation using Dijkstra's algorithm whose time complexity is $O(n_E + n_{V1} \log n_{V1})$. Here n_E denotes the number of vertex pairs visible to each other.

Algorithm 2: Dominating Relationship Check

Input : Obstacles O, data points p_i and p_{i+1}, query point q
Output: Whether q is in the dominant region of p_i over p_{i+1}, $\beta_\Delta(p_1, p_2)$
1 $\Psi \leftarrow \{v \mid v$ is an obstacle vertex $v \in \phi_\Delta(p_i, d_\Delta(p_i, p_{i+1}))v \in \phi_\Delta(p_{i+1}, d_\Delta(p_i, p_{i+1}))\}$
2 $\Phi \leftarrow \{o \mid o \in O$ at least one vertex of o is in $\Psi\}$
3 Compute the visibility graph on p_i and p_{i+1} with the obstacles in Φ
4 Compute $d_\Delta(q, p_i)$
5 Compute $d_\Delta(q, p_{i+1})$
6 return $d_\Delta(q, p_i) \leq d_\Delta(q, p_{i+1})$

6 Obstructed Integrated Safe Region

In this section we combine OSRD and OFR to form a region where the movement of q does not cause its kNN set to change. We call this region the *Obstructed Integrated Safe Region (OISR)*.

OSRD $\omega_\Delta(q_b, p_i, l)$ ($p_i \in L_{k+x}$) guarantees that p_i is nearer to q than any data point not in L_{k+x}. Thus, $\bigcap_{i=1}^{k} \omega_\Delta(q_b, p_i, l)$ guarantees that the first k data points in L_{k+x} are nearer to q than any point not in L_{k+x}. OFR $\eta_\Delta(L_{k+x})$ guarantees that the distance rank of the points in L_{k+x} does not change, and that the first k points in L_{k+x} are nearer to q than the $(k+1)^{th}$ to the $(k+x)^{th}$ points in L_{k+x}. Thus, the intersection of $\bigcap_{i=1}^{k} \omega_\Delta(q_b, p_i, l)$ and $\eta_\Delta(L_{k+x})$ guarantees that the first k points in L_{k+x} form the kNN set of q and this kNN set does not change by the movement of q. This intersection is the obstructed integrated safe region.

Definition 3 (Obstructed integrated safe region) The obstructed integrated safe region of L_{k+x}, denoted by $\Omega_\Delta(q_b, L_{k+x})$, is a region that, when q moves inside the region, the first k data points in L_{k+x} are always the kNN of q, and the distance ranks of these k points do not change. It is the intersection of $\bigcap_{i=1}^{k} \omega_\Delta(q_b, p_i, l)$ and $\eta_\Delta(L_{k+x})$. Formally,

$$\Omega_\Delta(q_b, L_{k+x}) = \eta_\Delta(L_{k+x}) \cap \bigcap_{i=1}^{k} \omega_\Delta(q_b, p_i, l). \tag{17}$$

An example is shown in Figure 10, where $k = 2$, $x = 1$, $L_3 = \langle p_1, p_2, p_3 \rangle$ and the query point is at q_b. The OISR $\Omega_\Delta(q_b, L_3)$ is formed by the intersection of $\omega_\Delta(q_b, p_1, l)$, $\omega_\Delta(q_b, p_2, l)$, $\beta_\Delta(p_1, p_2)$ and $\beta_\Delta(p_2, p_3)$, as shown by the dotted grey region.

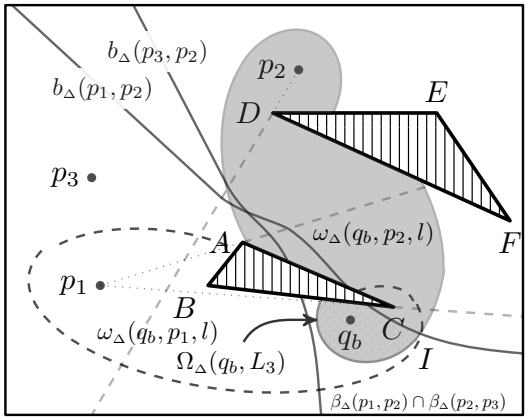


Fig. 10: Obstructed integrated safe region

Computation. In Equation 17, computing the intersection of multiple safe regions is too expensive. In what follows, we prove that $\eta_\Delta(L_{k+x})\cap\bigcap_{i=1}^{k}\omega_\Delta(q_b,p_i,l)=\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l)$. Then, we just need to compute one safe region for $\Omega_\Delta(q_b,L_{k+x})$.

Theorem 3 *Let $L_{k+x}=\langle p_1,p_2,...,p_{k+x}\rangle$ be a list of data points ordered by their distances to q_b. Then,*

$$\eta_\Delta(L_{k+x})\cap\bigcap_{i=1}^{k}\omega_\Delta(q_b,p_i,l)=\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l). \tag{18}$$

Proof (i) "$\eta_\Delta(L_{k+x})\cap\bigcap_{i=1}^{k}\omega_\Delta(q_b,p_i,l)\Rightarrow\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l)$" is straightforward since the left part of the equation is a subset of the right part.

(ii) "$\eta_\Delta(L_{k+x})\cap\bigcap_{i=1}^{k}\omega_\Delta(q_b,p_i,l)\Leftarrow\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l)$". Let t be a point in $\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l)$. Then t is in $\eta_\Delta(L_{k+x})$ and $d_\Delta(t,p_k)+d_\Delta(q_b,t)\le l$. Also, t in $\eta_\Delta(L_{k+x})$ guarantees that $d_\Delta(t,p_i)\le d_\Delta(t,p_k),\forall i\in[1..k-1]$. Thus, $d_\Delta(t,p_i)+d_\Delta(q_b,t)\le l,\forall i\in[1..k-1]$ and t is in $\bigcap_{i=1}^{k}\omega_\Delta(q_b,p_i,l)$.

As Figure 10 shows, the intersection of $\omega_\Delta(q_b,p_1,l)$, $\omega_\Delta(q_b,p_2,l)$, $\beta_\Delta(p_1,p_2)$ and $\beta_\Delta(p_2,p_3)$ is the same as that of $\omega_\Delta(q_b,p_2,l)$, $\beta_\Delta(p_1,p_2)$ and $\beta_\Delta(p_2,p_3)$.

Based on Theorem 3, the computation of $\Omega_\Delta(q_b,L_{k+x})$ is simplified to be: $\Omega_\Delta(q_b,L_{k+x})=\eta_\Delta(L_{k+x})\cap\omega_\Delta(q_b,p_k,l)$.

Obstructed M*k*NN algorithm. Since the OFR part of an OISR is not precomputed or stored but replaced by the dominating relationship check, the OISR cannot be precomputed or stored. Thus, the *obstructed MkNN algorithm* only uses the OISR conceptually to process the obstructed M*k*NN query as follows. When an obstructed M*k*NN query is issued and the query point q is at q_b, the algorithm first computes the list L_{k+x}. Then the algorithm starts the maintenance process. At every timestamp, the new position of q arrives. The algorithm applies Algorithm 2 to perform dominating relationship checks for the data points in L_{k+x} to see whether q is still in $\eta_\Delta(L_{k+x})$. If it is not, then the algorithm re-sorts the data points in L_{k+x} to determine the new kNN of q. Next, the algorithm checks whether q is in the OSRD of p_k (computes the OSRD if it has not been computed yet). If it is, then q is still in its current OISR and no further processing is required. Otherwise, the current OISR is deprecated and

Algorithm 3: Obstructed MkNN processing

```
Input  : Obstacles O, query point q, query parameter k and system parameter x
Output: Obstructed MkNN of q
1  while true do
2      Compute a list L_{k+x} of the k+x nearest data points of q
3      Compute the known region on L_{k+x}
4      foreach timestamp do
5          foreach p_i in L_{k+x}, i < k do
6              if not DRC(O, p_i, p_{i+1}, q) then
7                  Sort the data points in L_{k+x}
8                  q_b ← q
9                  ω_Δ(q_b, p_k, l) ← OSRDC(O, p_k, q_b)
10                 break
11         if q ∉ ω_Δ(q_b, p_k, l) then
12             Compute a list L_{k+x} of the k+x nearest data points of q
13             Compute the known region on L_{k+x}
14         Output the first k points in L_{k+x}
```

a new L_{k+x} list needs to be computed. The above process repeats and the obstructed MkNN set of q is generated continuously (i.e., at every timestamp).

Algorithm 3 summarizes the process.

7 Essential Auxiliary Set

In this section we discuss how to choose the x extra data points when we compute the $k+x$ data points for safe region computation. Our aim is to (i) obtain safe regions as large as possible, so that recomputation can be less frequently, and (ii) constrain the search space for the x extra data points, so that the cost of each recomputation is reduced. Note that the x extra data points do not have to be the current $(k+1)^{th}$ to $(k+x)^{th}$ nearest data points. This is because an extra data point only needs to be able to validate the current safe region. It does not have to be (and we cannot predict anyway) the next nearest data point if the current safe region becomes invalid.

We introduce a concept called the *essential auxiliary set (EAS)* to constrain our search of the x extra data points. First, we define the data points that will not affect the size of an OISR whether or not they are used as the extra data points. The remaining data points form the EAS, i.e., the EAS contains data points that may affect the size of the OISR, and the extra data points should only be chosen from the EAS. Then we present a way to compute the EAS efficiently.

7.1 Determination of Inessential Data Points

If a data point p does not affect the size of an OISR whether or not it is used as an extra data point, then we say p is *inessential* to the OISR and call it an *inessential data point*. Otherwise, we say that p is *essential* to the OISR and call it an *essential data point*.

The following lemma gives a basic requirement that an essential data point must satisfy. If a data point does not satisfy the following lemma, then it is an inessential data point.

Lemma 4 *If a data point p is essential to an OISR, then there exists a point q' in the OISR where p is its $(k+1)^{th}$ nearest neighbor.*

Proof Since we are considering the possible extra data points, p must not be one of the current kNNs. Let it be the current $(k+\alpha)^{th}$ nearest neighbor of the query point q, where $\alpha \geq 1$. If p is essential, then the OISR is different computed with or without it. This implies that when q moves, p would become one of the kNNs, which means the nearness rank of p would become less than k. Since the distance from q to any data point changes continuously, the nearness rank of p has to change continuously from the original $k+\alpha$ to less than k, which encloses the rank $k+1$. When p is the $(k+1)^{th}$ neighbor, the point where q is at is the point q'.

This lemma gives a way to determine whether p is essential. However, it is too difficult to compute if possible at all, because potentially we need to check every possible point q' to see if p is its $(k+1)^{th}$ nearest neighbor.

To simplify the determination process, we consider the bisectors between p and the current kNNs.

Lemma 5 *A data point p is inessential to an OISR if and only if the OISR does not overlap any bisector between p and a current kNN.*

Proof (i) "p is inessential $\Rightarrow$ the OISR does not overlap any bisector between p and a current kNN".

Suppose the OISR overlaps a bisector between p and a current kNN, denoted by r. Then when the query point q moves from one side of the bisector to the other side within the OISR, p would become nearer to q than r does. This violates the condition that p is inessential. Therefore, if p is inessential to the OISR then the OISR must not overlap the bisector between p and r.

(ii) "the OISR does not overlap any bisector between p and a current kNN $\Rightarrow$ p is inessential".

Suppose p is essential, then there exists a point q' satisfying Lemma 4. When the query point moves from q' to p along their shortest obstructed path, p will become the k^{th} nearest neighbor when the query point is at some point q''. At this moment, we have (1) q'' must be at the border of the OISR, since p is becoming one of the kNNs, and the current OISR will become invalid, and (2) there is a current kNN r that has the same obstructed distance to q'' as p does, since otherwise p would have already become one of the kNNs. As a result, the OISR must overlap the bisector between p and r.

Therefore, if the OISR does not overlap any bisector between p and a current kNN then p is inessential.

So far we have assumed the existence of an OISR for the computation of an inessential data point. However, we do not have an OISR until we have identify the EAS and further the x extra data points. To bypass this dilemma, we use a region that is guaranteed to enclose any OISR that is computed using the current kNNs. Since this region is larger than any of the OISRs, using it in Lemma 5 will guarantee no false positive in identifying the inessential data points.

Lemma 6 *Any OISR of a* k*NN set R resides in an order-k obstructed Voronoi cell w.r.t. R.*

Proof The correctness of the lemma is guaranteed by the definition of an order-k obstructed Voronoi cell [23].

Now we can use the order-k obstructed Voronoi cell to identify the inessential data points:

Theorem 4 *Given a* k*NN set R and a data point p, if for every data point r ∈ R, the obstructed bisector between r and p is not overlapped by the order-k obstructed Voronoi cell w.r.t. R, then p is inessential to any OISR of R.*

Proof By Lemma 6, any OISR of R is fully contained in the order-k Voronoi cell w.r.t. R. Therefore, If the bisectors are not overlapped by the order-k Voronoi cell, then they will not be overlapped by any of the OISRs. Then by Lemma 5, p is inessential to R.

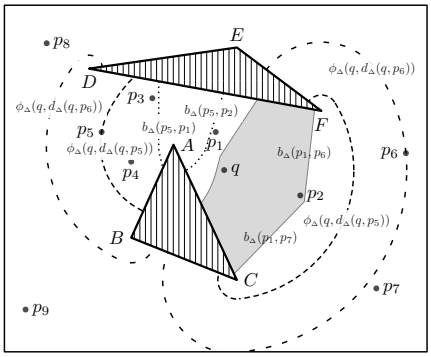


Fig. 11: Essential auxiliary set

Figure 11 gives an example, where $k = 2$, $x = 3$ and the query point is at q. The current kNN set $R = \{p_1, p_2\}$. Let 3 randomly chosen extra data points be $\{p_3, p_4, p_5\}$. The known region is enclosed by $\phi_\Delta(q, d_\Delta(q, p_5))$ (denoted by the densely dashed curve). The grey region is the order-2 Voronoi cell w.r.t. $\{p_1, p_2\}$. Then p_5 is actually inessential to R since its bisectors with p_1 and p_2 (denoted by the dotted curves) are not overlapped by the Voronoi cell. On the other hand, p_6 and p_7 are essential to R since the bisectors of p_1 with p_6 and p_7 contribute as parts of the Voronoi cell boundary.

The aim of EAS is to eliminate points like p_5 from being considered as the extra data points, so that we could have more effective extra data points and a larger known region (e.g., $\phi_\Delta(q, d_\Delta(q, p_6))$, denoted by the loosely dashed curve.

7.2 Computation of the Essential Auxiliary Set

Theorem 4 gives a more computable definition of the inessential data points. However, the order-k Voronoi cell involved in the theorem is impracticable to be used due to its complexity [23]. In this subsection, we use the *obstructed Delaunay triangulation (ODT)* to replace the order-k Voronoi cell, which can be pre-computed easily for a given set of objects and obstacles without sacrificing the algorithm correctness. We also compute the EAS based on the obstructed Delaunay triangulations.

Definition 4 The *Delaunay triangulation* for a set P of points is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$ [23].

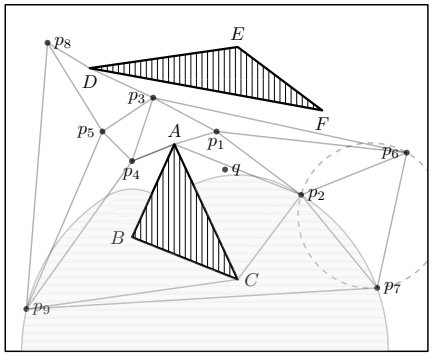


Fig. 12: Obstructed Delaunay triangulation

When obstacles are involved, the *obstructed Delaunay triangulation (ODT)* is defined by substituting the edges of the triangles by the obstructed shortest paths, and the circumcircles by "circum-obstructed-disks". Figure 12 gives an example, where the dashed circle indicates the circum-obstructed-disk w.r.t. p_2, p_6 and p_7, and the gray obstructed disk indicates the circum-obstructed-disk w.r.t. p_2, p_7 and p_9.

Next, we generalize Theorem 4 based on the ODT.

Lemma 7 *Given a* 1*NN data point r, a data point p is inessential to r if p and r are not directly linked by an edge in the ODT.*

Proof We need to prove that if p and r are directly linked by an edge in the ODT then the bisector between r and p is not overlapped the Voronoi cell of r.

Suppose the bisector is overlapped by the Voronoi cell. Let a moving point d be in the Voronoi cell, then we can draw an obstructed disk $\phi_\Delta(d, d_\Delta(d,p))$. This disk encloses r and p on the border and no object inside, since no object in the Voronoi cell of r can be nearer to d than r. Next, we move d along the bisector until $\phi_\Delta(d, d_\Delta(d,p))$ encounters another object o on its border. At this moment, we have p, r and o all on the border of an obstructed disk and no other object inside, which means that p, r and o define a triangle in the ODT, and we encounter a contradiction that p and r are directly linked by an edge in the ODT.

Lemma 8 *Given a* k*NN set R, if for data point r ∈ R, a data point p is not directly linked by an edge to r in the ODT ignoring all other data points in R, then p is inessential to R.*

Proof Every Delaunay triangulation has a corresponding dual Voronoi diagram [23] . We prove this lemma through the properties of the Voronoi diagram. The order-k Voronoi cell for a set R is intersected by the order-1 Voronoi cell of each data point $r \in R$ computed when the other objects in R are ignored. If p is not directly linked to r, then by Lemma 7 and Lemma 5, the bisector between p and r is not overlapped by the Voronoi cell of r. As a result, the bisectors between p and any data point in R is not overlapped by the order-k Voronoi cell of R. Therefore, p is inessential to R.

Lemma 8 needs different ODT's, which are constructed by ignoring different sets of objects. The following lemma allows us to eliminate the need of these different ODT's.

Lemma 9 *Given the ODT w.r.t. a data point set R (denoted by ODT_R), and the ODT w.r.t. a subset P of R (denoted by ODT_P), for data point p ∈ P, if p is not directly linked by an edge*

to any data point in $R \backslash P$ in ODT_R, then the edges linked to p are exactly the same in both ODT's.

The correctness of Lemma 9 can be proved straightforwardly based on the the unobstructed version of the lemma in [23], and hence the proof is omitted. Based on the lemmas above, we have the following conclusion.

Theorem 5 *Given a* k*NN set R, a data point p is inessential to R if there is no edge in the obstructed Delaunay triangulation between p and any data point in R.*

Proof The correctness of the theorem is guaranteed by the lemmas above straightforwardly.

Theorem 5 suggests that data points not directly linked to the data points in R are inessential and can be pruned from consideration when choosing the extra data points. Therefore, we can keep the data point that linked to the data points in R as the EAS. Formally:

Definition 5 The *essential auxiliary set (EAS)* for a kNN set R is a set of data points that are directly linked to at least one of the data points in R in the obstructed Delaunay triangulation.

The definition suggests an efficient way to construct the EAS when the extra data points are needed. Note that the extra data points are needed after the kNN retrieval and before constructing the OISR. Therefore, the EAS can be constructed as follows:

1. In the initialization phase, pre-compute the obstructed Delaunay triangulation based on all data points and obstacles, by using the technique presented in [4].
2. For each data point p, store a set of references to the other data points, denoted by L_p, that are directly linked to p by the edges in the Delaunay triangulation.
3. When processing a MkNN query, after the retrieval of the current kNN set R, compute the EAS as the following union:

$$EAS = (\bigcup_{r \in R} L_r) \backslash R. \tag{19}$$

4. Sort EAS by based on the distance of the data points to the query point and take the x nearest ones as the x extra data points for safe region computation.

Note that the EAS may contain false positives, i.e., inessential data points, since a data point directly linked to the data points in R can still be inessential to R. However, we have the following theorem that guarantee that on average the number of data points in EAS is constrained, i.e., less than or equal to $6k$.

Theorem 6 *The average number of data points in an EAS is less than or equal to* $6k$.

Proof In the dual Voronoi diagram of a Delaunay triangulation, an order-k Voronoi cell is constructed by k order-1 Voronoi cells. Since on average each Voronoi cell has 6 neighbors [23], for a set R of k data points, it has no more than $6k$ neighbors in total.

8 Other Types of Obstacles

Until now our discussion has assumed convex polygons with non-zero extents as the obstacles. In this section we discuss how our techniques can be applied on other types of obstacles.

Concave polygons with non-zero extents. In this case we can simply divide each obstacle into multiple convex polygons. Then our techniques can be applied straightforwardly.

Line segments. A typical use case of line segments as obstacles is shopping recommendation in large shopping malls. As shown in Figure 13, line segments $\overline{AB}, \overline{BC}, ..., \overline{TA}$ form the outline of a shopping mall. A customer in the mall is represented by q. She may issue a query to the shopping recommendation server about the types of shops she is interested in. Then the server can identify the relevant shops (as the data points) and make continuous recommendation based on the distance between the customer and the shops as the customer is moving. In this case, the walls (i.e., the line segments) need to be considered when computing the distance between the customer and a shop. They form the obstacle set and their endpoints form the obstacle vertex set. Then our techniques can be applied to process the obstructed M*k*NN queries for this case.

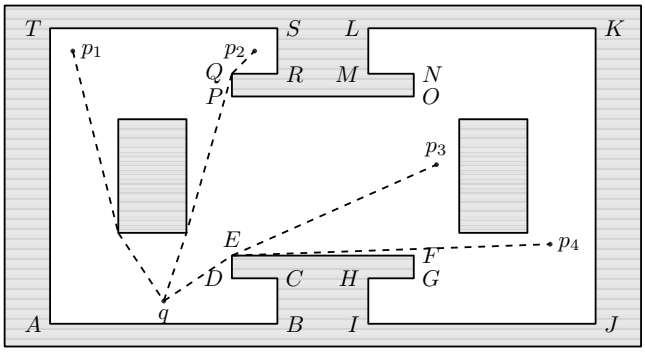


Fig. 13: Line segments as obstacles

9 Experiments

In this section, we present a detailed performance study of our obstructed M*k*NN algorithm. We first describe the experimental settings in Section 9.1, then we evaluate our proposed algorithm under different values of the algorithm parameter x in Section 9.2. We compare our algorithm with a baseline algorithm in Section 9.3. In the experiments we measure the number of page accesses, the communication cost (by counting the number of safe region recomputation), and the average response time.

9.1 Experimental Setup

All experiments were conducted on a desktop computer with a 2.4 GHz Intel i5 CPU and 8 GB main memory.

Datasets. We use real datasets as the obstacles and generate synthetic data as the data points and the query point. The real datasets used are the *Hypsography* dataset and the *Census Blocks* dataset from the R-tree Portal[1]. The *Hypsography* dataset contains 76,999 MBRs of hypsography data from Germany and the *Census Blocks* dataset contains 556,696 MBRs of census blocks from the US. We map these MBRs to a data domain of 200000 $\times$ 200000 units.

We generate two types of synthetic datasets around the obstacles: (i) uniform dataset (denoted by "U"), where the data points follow uniform distribution; (ii) Zipfian dataset (denoted by "Z"), where the data points follow Zipfian distribution with $\alpha = 0.85$ as the skew coefficient.

We use the R*-tree [1] to index the obstacles and the data points independently, where the page size is set to be 2KB and the buffer size is set to be 16 pages.

We generate two types of query point trajectories: (i) *random (denoted by "R")*, where the movement of the query point follows the Random Waypoint model; (ii) *directional (denoted by "D")*, where the query point moves in straight lines until reaching an obstacle and the movement is reflected by the obstacle.

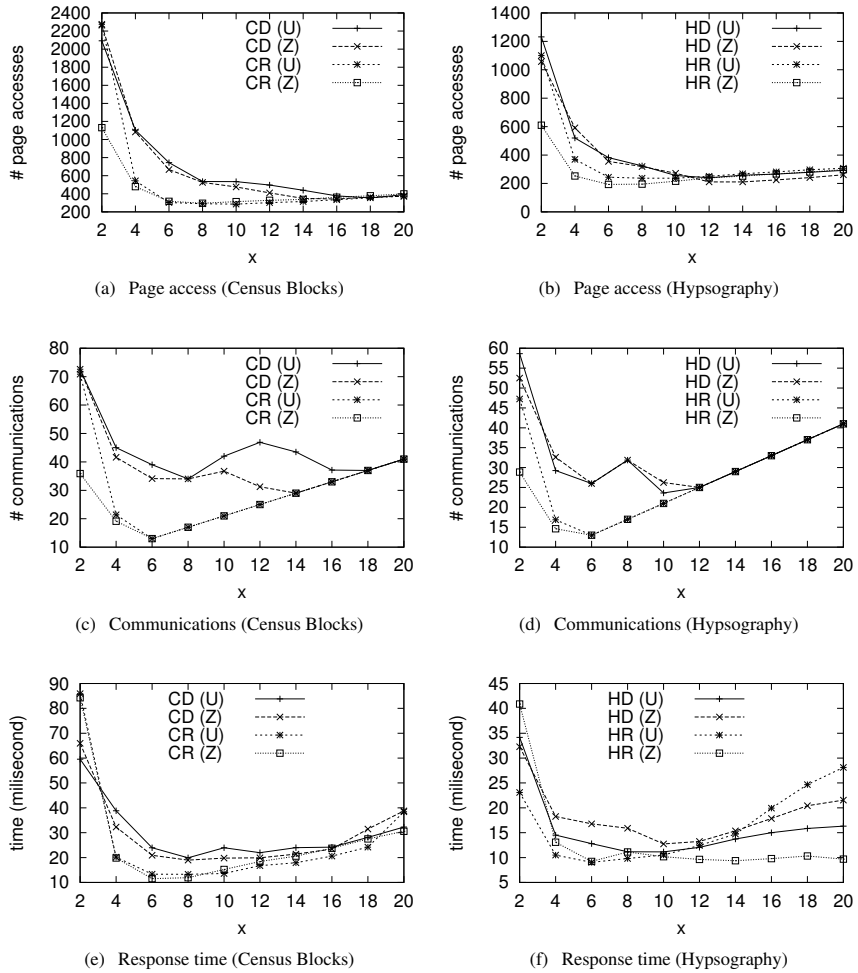
Parameters. To evaluate the algorithms under various settings, we vary the value of x from 2 to 20, the value of k from 5 to 50, the value of $|P|/|O|$ from 0.1 to 10, and the query point moving speed v_q from 100 to 500 units per timestamp. By default we set x at 10, k at 5, $|P|/|O|$ at 5 and v_q at 250. We run each experiment for 200 timestamps and report the average cost per timestamp.

Baseline algorithm. In the comparative study we adapt the *Continuous Obstructed kNN (COkNN) algorithm* [9] as the baseline algorithm. As discussed in Section 2.3, this algorithm assumes predefined linear trajectories for the query object. To use it in our problem, we treat each short trajectory segment from the last timestamp to the current timestamp as a predefined trajectory and feed it into the algorithm. This means the CO*k*NN algorithm will be run at each timestamp.

9.2 Effect of x

We maintain x extra nearest neighbors chosen based on the essential auxiliary set as a cache to reduce the frequency of safe region recomputation. In this set of experiments we evaluate the algorithm performance when the value of x is varied to choose the best value of x. As shown in Figure 14, when x increases from 2 to 20, the I/O cost of query processing decreases. The reason is that when x becomes larger, the obstructed known region becomes larger and hence the valid periods of the safe regions become longer. The frequency of safe region recomputation becomes smaller and thus, the number of page accesses to retrieve the positions of the data points and the obstacles as well as the communication cost to report the updated *k*NN set decrease. Meanwhile, the computation cost to maintain the safe regions of $k+x$ data points increases as x increases. As a result, the average response time increases when the value of x becomes too large. As shown in Figure 14(e) and (f), different data sets have different best x values. Overall, our proposed algorithm performs best when $x = 10$. Therefore, we will use this value in our algorithm in the following experiments.

[1] `http://www.chorochronos.org/`



(a) Page access (Census Blocks)
(b) Page access (Hypsography)
(c) Communications (Census Blocks)
(d) Communications (Hypsography)
(e) Response time (Census Blocks)
(f) Response time (Hypsography)

Fig. 14: Effect of x

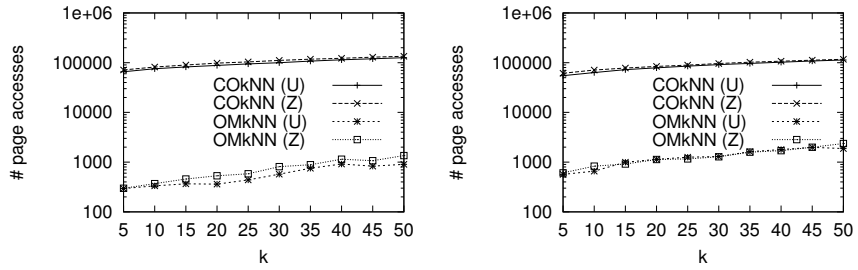
9.3 Comparative Study

In this subsection we compare our obstructed M*k*NN algorithm with a baseline algorithm COkNN [9]. In the result figures we denote our algorithm by "OMkNN" while the baseline algorithm by "COkNN".

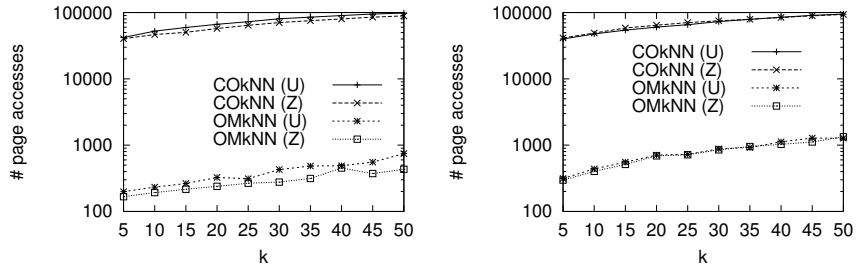
9.3.1 Varying the Query Parameter k

Figure 15 to Figure 17 show the comparative performance when the value of k is varied. From the figures we can see that our algorithm outperforms COkNN in the three measurements by up to two orders of magnitude (please note the logarithmic scale).

Figure 15 shows the numbers of page accesses. Our algorithm has much smaller numbers because our safe region based algorithm keeps the number of *k*NN recomputation low and hence it does not require to access the data point as frequently as the COkNN does. The

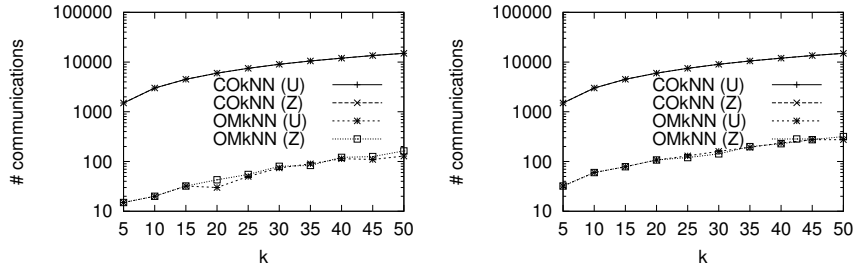


(a) Census block and Random (b) Census block and Directional

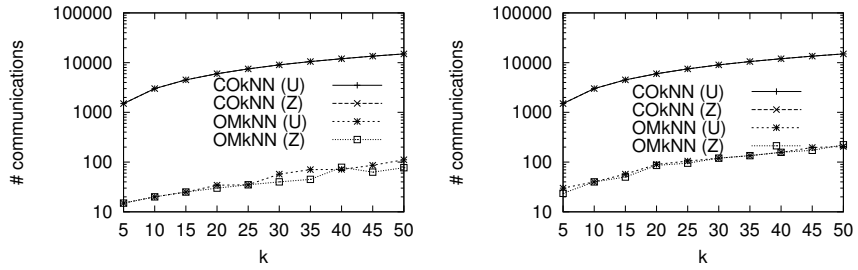


(c) Hysography and Random (d) Hysography and Directional

Fig. 15: Page accesses vs. k

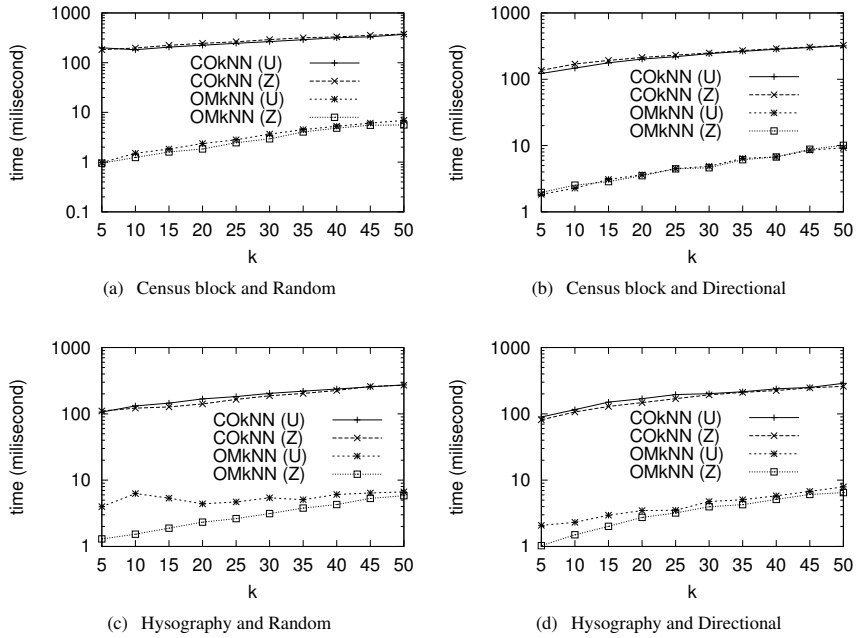


(a) Census block and Random (b) Census block and Directional



(c) Hysography and Random (d) Hysography and Directional

Fig. 16: Communications vs. of k



(a) Census block and Random

(b) Census block and Directional

(c) Hysography and Random

(d) Hysography and Directional

Fig. 17: Response time vs. of k

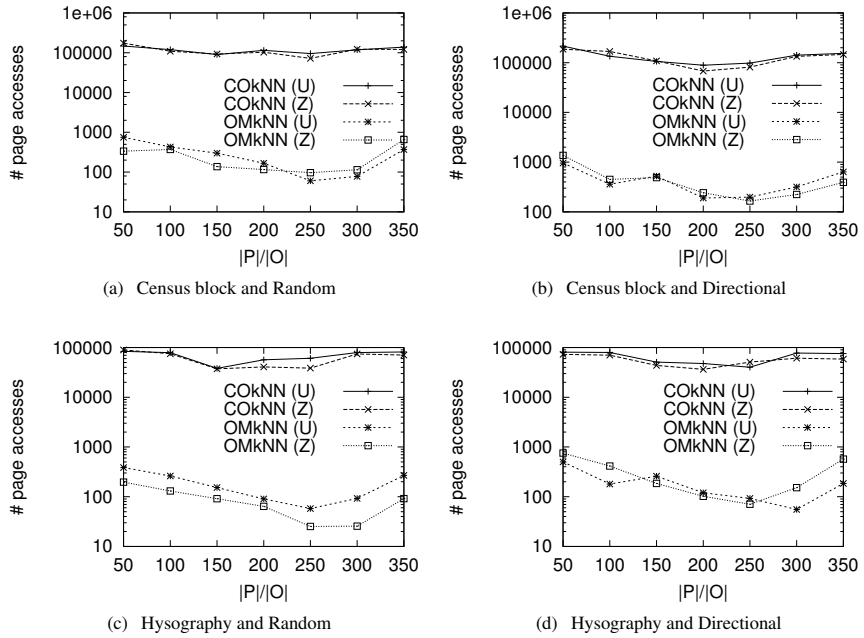
same reason applies and thus the communication cost of our algorithm is also much smaller than that of COkNN (cf. Figure 16).

The above two effect together result in a smaller average response time for our algorithm. As Figure 17 shows, for COkNN, the average response time is almost one second per timestamp, which means COkNN is too slow to generate valid query results for more than just 1 query at the same time. On the contrary, the response time of our algorithm is below 10 milliseconds for most cases, which means our algorithm can provide timely query results for tens of queries simultaneously (Please note that the experiments were conducted on an average desktop computer rather than a high performance server machine). This demonstrates the scalability of our algorithm.

9.3.2 Varying Data Point-Obstacle Ratio $|P|/|O|$

Next we compare our algorithm with COkNN when the ratio of the number of data points over the number of obstacles ($|P|/|O|$) is varied.

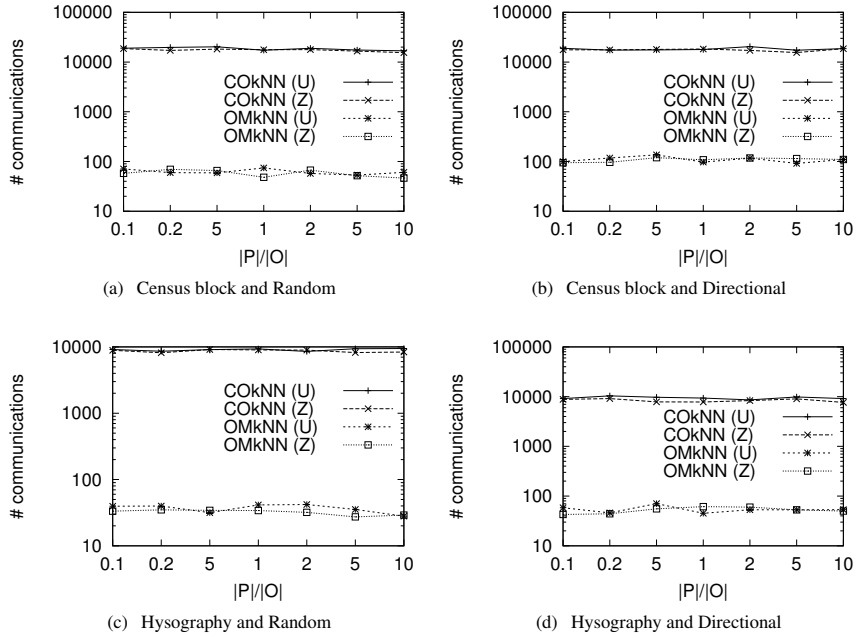
Figure 18 to Figure 20 give the result. Our algorithm again outperforms COkNN by more than one order of magnitude in the three measurements. We can see from Figure 20 that as the ratio grows, the average response time of our algorithm first drops and then increases again. This is because, when $|P|/|O|$ is small, the data points are sparse and the obstructed known region is large and it encloses more obstacles. As a result, the maintenance cost of the safe regions is high. As $|P|/|O|$ grows, the density of data points increases, the size of the obstructed know region drops and it encloses less obstacles. Thus, the maintenance cost of the safe regions drops. However, when the size of the obstructed know region drops the frequency of safe region recomputation also increases. This effect becomes the dominating effect and the costs of query processing grow again when $|P|/|O|$ is larger than 2.



(a) Census block and Random (b) Census block and Directional

(c) Hysography and Random (d) Hysography and Directional

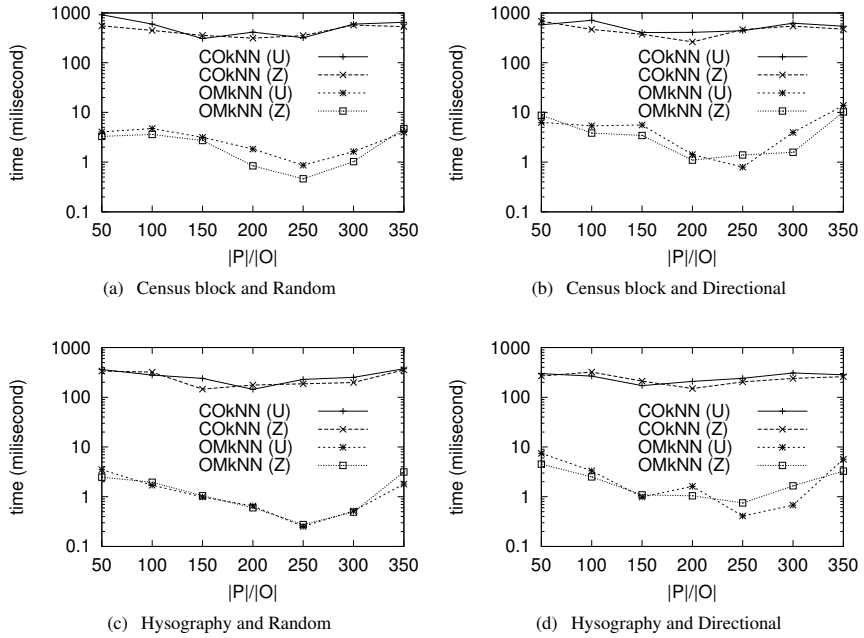
Fig. 18: Page accesses vs. $|P| / |O|$



(a) Census block and Random (b) Census block and Directional

(c) Hysography and Random (d) Hysography and Directional

Fig. 19: Communications vs. $|P| / |O|$



(a) Census block and Random (b) Census block and Directional

(c) Hysography and Random (d) Hysography and Directional

Fig. 20: Response time vs. $|P| / |O|$

Note that the communication cost of COkNN stays unchanged when $|P|/|O|$ is varied (cf. Figure 19). This is because COkNN has to be re-run and sends the new kNN set at every timestamp regardless of the value of $|P|/|O|$. The communication cost is only related to the size of the kNN set, which is decided by k and x.

9.3.3 Varying Query Point Speed v_q

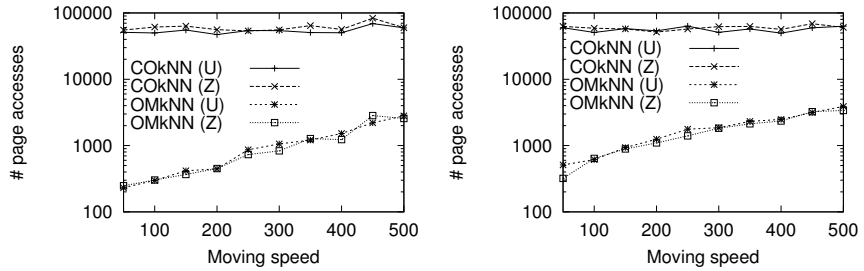
We also vary the query point speed v_q. Figure 23 shows the response time. OMkNN again outperforms COkNN significantly. It can provide query answers within 0.01 seconds when v_q is 500 units per timestamp. Assume that the 200000×200000 data space represents a 1000km $\times$ 1000km area. Then OMkNN provides query answers within 0.01 seconds when v_q is 180km/h, which is fast enough for most daily life applications. This shows the applicability of OMkNN.

The comparative performance in I/O cost and communication cost is similar to that of previous experiments.

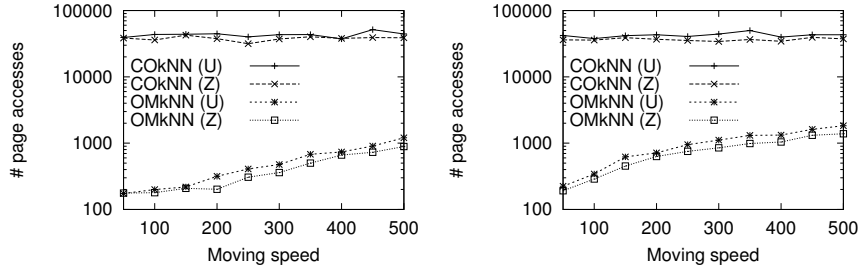
In summary, our obstructed MkNN algorithm outperforms the adapted baseline algorithm COkNN by up to two orders of magnitude in computation and communication costs under various settings.

10 Conclusion

In this paper we studied the problem of processing the obstructed MkNN query, i.e., the kNN query for a moving query point in space with obstacles. We proposed a safe region based algorithm to process the query. The safe region used is a combination of two types of

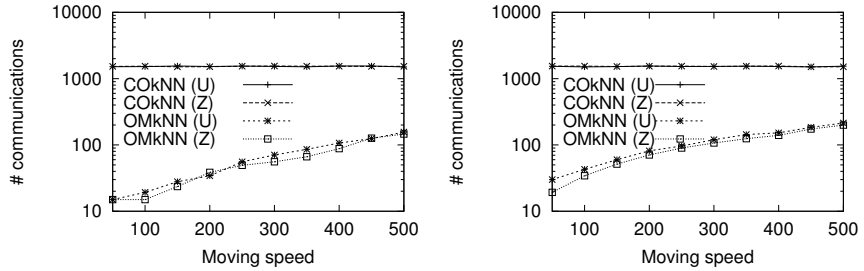


(a) Census block and Random (b) Census block and Directional

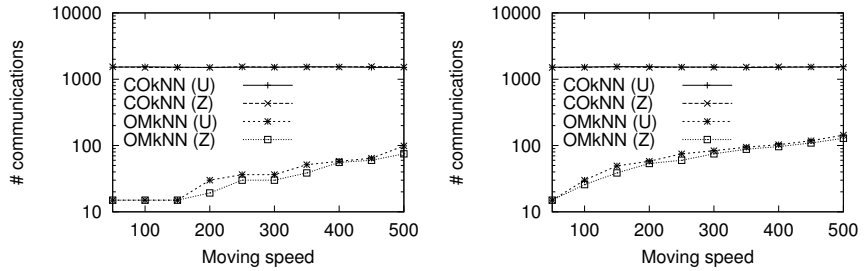


(c) Hysography and Random (d) Hysography and Directional

Fig. 21: Page accesses vs. v_q

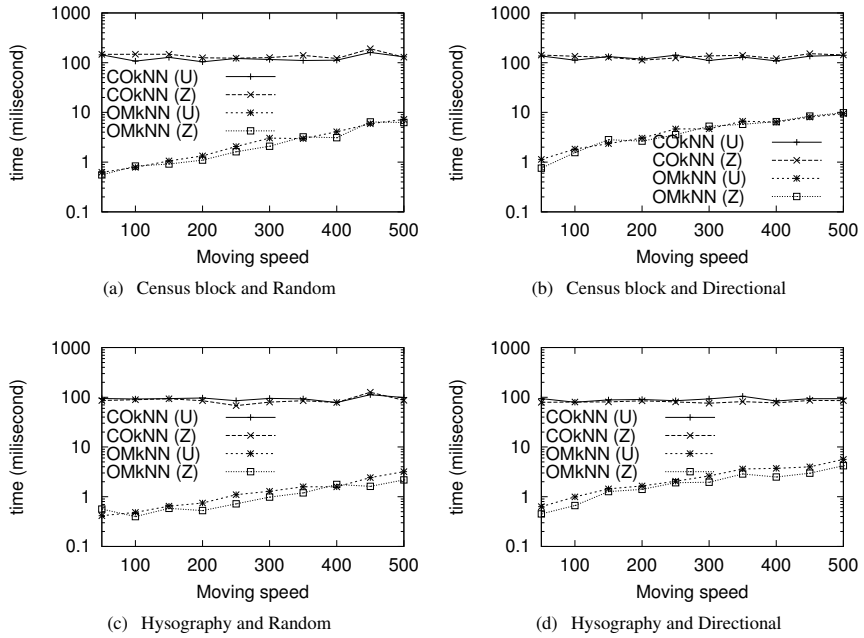


(a) Census block and Random (b) Census block and Directional



(c) Hysography and Random (d) Hysography and Directional

Fig. 22: Communications vs. v_q



(a) Census block and Random

(b) Census block and Directional

(c) Hysography and Random

(d) Hysography and Directional

Fig. 23: Response time vs. v_q

regions, i.e., the obstructed safe region w.r.t a data point (OSRD) and the obstructed fixed-rank region (OFR). The OSRD guarantees that the objects in the *k*NN set do not change, while the OFR guarantees that the closeness order of the objects in the *k*NN set to the query object also does not change. Together, they guarantee that, as long as the query object stays in their intersection, the query result does not need recomputation. Using these two types of regions, we obtained an efficient algorithm to process the M*k*NN query. We further optimized the algorithm through the essential auxiliary set which help determine the possible new *k*NNs when the *k*NN set needs to be updated. As the experiments show, our algorithm outperforms an adapted baseline algorithm by up to two orders of magnitude under various settings.

Compared with an earlier conference version of the paper [16] where a preliminary query processing algorithm was proposed, this journal article significantly enhanced the query processing efficiency by introducing the obstructed disk and the essential auxiliary set, which reduce the number of obstacle vertices and objects checked in safe region computation. We renewed the experiments comparing the enhanced algorithm with a more advanced baseline algorithm [9]. In addition, more types of obstacles are considered, and detailed theoretical foundations are provided, including formal definitions and proofs to the key techniques and complexity analysis.

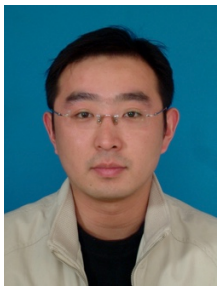
Acknowledgements This work is supported by the National Basic Research Program of China under Grant No. 2012CB316201, the National Natural Science Foundation of China under Grant No. 61300021, 61033007 and 61472071, Australian Research Council (ARC) Discovery Project DP130104587, Australian Research Council (ARC) Future Fellowships Project FT120100832, the Fundamental Research Funds for the Central Universities of China No. N120304003 and N130404010, National Key Technology R&D Program of China 2012BAK24B01 and China Scholarship Council.

References

1. Beckmann N, Kriegel H, Schneider R, Seeger B (1990) The r*-tree: an efficient and robust access method for points and rectangles. SIGMOD Record 19(2):322–331
2. Benetis R, Jensen S, Karčiauskas G, čaltenis S (2006) Nearest and reverse nearest neighbor queries for moving objects. VLDB Journal 15(3):229–249
3. Cheema MA, Lin X, Zhang Y, Wang W, Zhang W (2009) Lazy updates: an efficient technique to continuously monitoring reverse knn. PVLDB 2(1):1138–1149
4. Chew LP (1989) Constrained delaunay triangulations. Algorithmica 4(1-4):97–108
5. De Berg M, Gudmundsson J, Hammar M, Overmars M (2003) On r-trees with low query complexity. Computational Geometry 24(3):179–195
6. De Berg M, Cheong O, Van Kreveld M, Overmars M (2008) Computational geometry: algorithms and applications. Springer
7. Dijkstra E (1959) A note on two problems in connexion with graphs. Numerische mathematik 1(1):269–271
8. Eunus Ali M, Zhang R, Tanin E, Kulik L (2008) A motion-aware approach to continuous retrieval of 3d objects. In: ICDE, pp 843–852
9. Gao Y, Zheng B (2009) Continuous obstructed nearest neighbor queries in spatial databases. In: SIGMOD, pp 577–590
10. Gao Y, Yang J, Chen G, Zheng B, Chen C (2011) On efficient obstructed reverse nearest neighbor query processing. In: ACM SIGSPATIAL, pp 191–200
11. Gao Y, Zheng B, Chen G, Li Q, Guo X (2011) Continuous visible nearest neighbor query processing in spatial databases. VLDB Journal 20(3):371–396
12. Hsueh YL, Zimmermann R, Wang H, Ku WS (2007) Partition-based lazy updates for continuous queries over moving objects. In: GIS, pp 1–8
13. Hu H, Xu J, Lee DL (2005) A generic framework for monitoring continuous spatial queries over moving objects. In: SIGMOD, pp 479–490
14. Jagadish HV, Ooi BC, Tan K, Yu C, Zhang R (2005) idistance: An adaptive b^+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
15. Kolahdouzan M, Shahabi C (2004) Voronoi-based k nearest neighbor search for spatial network databases. In: Very Large Ddata Bases, pp 840–851
16. Li C, Gu Y, Li F, Chen M (2010) Moving k-nearest neighbor query over obstructed regions. In: Asia-Pacific Web Conference, pp 29–35
17. Li C, Gu Y, Yu G, Li F (2011) wneighbors: a method for finding k nearest neighbors in weighted regions. In: DASFAA, Springer, pp 134–148
18. Li C, Gu Y, Qi J, Yu G, Zhang R, Yi W (2014) Processing moving knn queries using influential neighbor sets. Proceedings of the VLDB Endowment 8(2):113–124
19. Mokbel MF, Aref WG (2008) Sole: scalable on-line execution of continuous queries on spatio-temporal data streams. VLDB Journal 17(5):971–995
20. Mokbel MF, Xiong X, Aref WG (2004) Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD, pp 623–634
21. Mouratidis K, Papadias D, Bakiras S, Tao Y (2005) A threshold-based algorithm for continuous monitoring of k nearest neighbors. TKDE 17:1451–1464
22. Nutanong S, Zhang R, Tanin E, Kulik L (2008) The v*diagram: A query dependent approach to moving knn queries. In: Very Large Ddata Bases, pp 1095–1106
23. Okabe A, Boots B, Sugihara K, Chiu SN (2000) Spatial Tessellations. John Wiley & Sons, Inc
24. Preparata F, Shamos M (1985) Computational geometry: an introduction. Springer

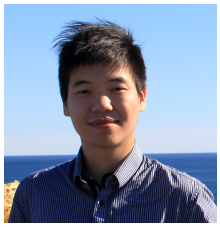
25. Song Z, Roussopoulos N (2001) K-nearest neighbor search for moving query point. In: SSTD, pp 79–96
26. Tao Y, Papadias D (2002) Time-parameterized queries in spatio-temporal databases. In: SIGMOD, pp 334–345
27. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: Very Large Data Bases, pp 287–298
28. Šaltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. In: SIGMOD, pp 331–342
29. Wang Y, Zhang R, Xu C, Qi J, Gu Y, Yu G (2014) Continuous visible k nearest neighbor query on moving objects. Inf Syst 44:1–21
30. Ward PG, He Z, Zhang R, Qi J (2014) Real-time continuous intersection joins over large sets of moving objects using graphic processing units. The VLDB Journal pp 1–21
31. Xia C, Hsu D, Tung AK (2004) A fast filter for obstructed nearest neighbor queries. In: Key Technologies for Data Management, Springer, pp 203–215
32. Xia T, Zhang D (2006) Continuous reverse nearest neighbor monitoring. In: ICDE, pp 77–86
33. Xiong X, Mokbel MF, Aref WG (2005) Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: ICDE, pp 643–654
34. Yu X, Pu KQ, Koudas N (2005) Monitoring k-nearest neighbor queries over moving objects. In: ICDE, pp 631–642
35. Zhang J, Zhu M, Papadias D, Tao Y, Lee DL (2003) Location-based spatial queries. In: SIGMOD, pp 443–454
36. Zhang J, Papadias D, Mouratidis K, Zhu M (2004) Spatial queries in the presence of obstacles. In: International Conference on Extending Database Technology, pp 366–384
37. Zhang R, Jagadish HV, Dai BT, Ramamohanarao K (2010) Optimized algorithms for predictive range and knn queries on moving objects. Information Systems 35(8):911–932
38. Zhang R, Qi J, Lin D, Wang W, Wong RCW (2012) A highly optimized algorithm for continuous intersection join queries over moving objects. VLDB Journal 21(4):561–586

Author Biographies



Chuanwen Li received his B.E., M.E., and Ph.D degree in Computer Software and Theory from Northeastern University of China, in 2005, 2008 and 2011, respectively. Currently, he is a lecture in Northeastern University, China. His current research interests include spatial data management and realtime scheduling. He is a member of CCF.

Yu Gu received his B.E., M.E., and Ph.D degree in Computer Software and Theory from Northeastern University of China, in 2004, 2007 and 2010, respectively. Currently, he is an associate professor in Northeastern University, China. His current research interests include spatial data management and graph data management. He is a senior member of CCF.



Jianzhong Qi received his Ph.D degree in the Department of Computing and Information Systems at the University of Melbourne, in 2014. Currently, he is a research fellow at the University of Melbourne. He has been an intern at Toshiba China R&D Center and Microsoft Redmond in 2009 and 2013, respectively. His research interests include spatial databases, location-based social networks, information extraction and web data mining.

Rui Zhang is an Associate Professor and Reader in the Department of Computing and Information Systems at the University of Melbourne. He obtained his bachelor's degree from Tsinghua University in 2001 and his PhD from National University of Singapore in 2006. He has been a visiting research scientist at AT&T labs-research in New Jersey and at Microsoft Research in Redmond, Washington. Since January 2007, he has been a faculty member in the Department of Computing and Information Systems at the University of Melbourne. Recently, he has been a visiting researcher at Microsoft Research Asia in Beijing regularly collaborating on his ARC Future Fellowship project. His research interest is data and information management in general, particularly in areas of indexing techniques, moving object management, web services, data streams and sequence databases.

Ge Yu received his Ph.D degree in Computer Science from Kyushu University of Japan in 1996. He is now a professor in Northeastern University of China. His research interests include distributed and parallel database, data integration, graph data management, etc. He is a member of the IEEE Computer Society, the ACM, and the CCF.