World Wide Web (2016) 19:653–677
DOI 10.1007/s11280-015-0351-3

Efficient moving *k* nearest neighbor queries over line segment objects

Yu Gu[1] · Hui Zhang[1] · Zhigang Wang[1] · Ge Yu[1]

Received: 8 April 2014 / Revised: 15 March 2015 /
Accepted: 20 April 2015 / Published online: 5 May 2015

Abstract The growing need for location based services motivates the moving *k* nearest neighbor query (M*k*NN), which requires to find the *k* nearest neighbors of a moving query point continuously. In most existing solutions, data objects are abstracted as points. However, lots of real-world data objects, such as roads, rivers or pipelines, should be reasonably modeled as line segments or polyline segments. In this paper, we present LV*-Diagram to handle M*k*NN queries over *line segment* data objects. LV*-Diagram dynamically constructs a safe region. The query results remain unchanged if the query point is in the safe region, and hence, the computation cost of the server is greatly reduced. Experimental results show that our approach significantly outperforms the baseline method w.r.t. CPU load, I/O, and communication costs.

Keywords k nearest neighbor · Line segments · Continuous queries · Spatial queries

1 Introduction

The popularization of GPS-equipped mobile devices and web services have motivated various queries and analysis over spatial data [15, 21]. Among these, a typical class of queries

✉ Yu Gu
guyu@ise.neu.edu.cn

Hui Zhang
zhanghui@ise.neu.edu.cn

Zhigang Wang
wangzhigang@ise.neu.edu.cn

Ge Yu
yuge@ise.neu.edu.cn

[1] Northeastern University, Shenyang, People's Republic of China

called the *Moving k Nearest Neighbor* (M*k*NN) query [16, 27] have been widely studied, where a moving point continuously issues queries to find its *k* nearest neighbors. In particular, the *Moving k Nearest Neighbor* (M*k*NN) query are regarded as the most typical query type for location based services. For example, a tourist walking in a town looks for the nearest hotels, or a driver desired to continuously seek the nearest gas station. Essentially, M*k*NN queries are location-based continuous spatial queries, where the query results may vary depending on the current location of the query point due to its moving nature.

Generally, such applications are implemented by a Client/Server architecture, where the mobile devices simply issue queries and receive results, while the server focuses on the computation of results. Typically, users expect a real-time response, which raises great challenges in terms of the computation capacity of the server and the communication efficiency. Therefore, existing work on M*k*NN mainly centers on optimization techniques to reduce the server load and communication cost. A straightforward solution to process M*k*NN queries is the *sampling*-based approach [23], in which an M*k*NN query is processed as a sequence of *k*NN sub-queries at sampled locations. In order to provide continuous and correct query results, a high sampling rate is required, which exacerbates the load on both the server and the communication channel.

To overcome the shortcomings of *sampling*-based approaches, the *safe-region*-based schemes have been intensively investigated [18, 25]. As long as the query point stays in a safe region, the results remain unchanged. Thus, a new query is issued only when the query point exits the current safe region (i.e., the query results may change). *safe-region*-based solutions can provide accurate results without the frequent *sampling*, and hence reduce the load on both servers and communication channels. Additionally, the V*-Diagram [18] can compute the safe regions incrementally with local information, which further alleviates the server load.

Although a lot of efforts have been devoted to the efficient M*k*NN query processing, most existing methods abstract the data object as a point in the space. However, this abstraction may not be reasonable in real world where data objects could be roads, rivers, pipelines, etc. In such cases, data objects should be properly modeled as line segments or polyline segments instead of points. Also, compared to points, line segments can be more properly leveraged to approximately represent some objects with a certain depth in scenarios such as buildings, islands, etc. Note that in traditional spatial databases, line segments have been highly recognized as fundamental spatial information representations and *k*NN queries over line segments objects can also be handled using R-tree and its variants. Similarly, compared to such snap-shot queries, the (M*k*NN) queries can better push the answers to users in a more real-time manner for better pre-warning or decision supports. For example, for a wild explorer, continuously monitoring the nearest exiting roads or the nearest rivers can be helpful to his/her safety. Also, in a military exercise, moving soldiers need to keep being aware of the nearest trenches or shielding walls for quick defence. In these scenarios, the (M*k*NN) queries over line segments-like objects are preferred while the traditional snap-shot solutions are quite inefficient for the continuous query demands.

In this paper, we focus on the problem of LM*k*NN (line segment oriented moving k nearest neighbor) query with no predefined query trajectory. Specifically, LM*k*NN query continuously evaluates order-sensitive *k* nearest line segment (or poly line segment) neighbors while the query point moves freely in space, and returns to the user once a different *k*NN result is detected. Here, order-sensitive means we return a *k*NN sequence with the objects ordered instead of a simple *k*NN set, and thus different orders in the same *k*NN set will lead to different *k*NN results.

Our basic idea is inspired by V*-Diagram, the state-of-the-art method for the MkNN query with no predefined query trajectory in space. V*-Diagram is a safe region based technique. Its integrated safe region (ISR) is formed by two types of regions, the safe region w.r.t. a data point and the fixed-rank region(FRR). Our work is not a trivial extension of V*-Diagram because our work requires exploiting geometric properties entailed by line segments and polyline segments, which bring in significant difficulties in defining the safe regions and judging whether an object is in ISR. Specifically, constructing the line segment oriented safe region and implementing MkNN searching based on such irregular region face several typical challenges.

(i) Our fixed-rank region (FRR) is the closed region obtained by intersecting the (n - 1) bisectors of any two adjacent line segment objects in the rank list which is different from that in V*-Diagram. Furthermore, the whole data space is partitioned into multiple subspaces, and computing the bisector of any two objects must be distinguished according to the distance pattern of each subspaces. There are many kinds of bisectors in different subspaces like perpendicular bi-sector, parabola and angle bisector. Furthermore, the integrated safe-region is also different. Because the safe region with regard to an object in the original V*-Diagram is actually an oval. But for the case of line segment-oriented safe region, it is much more complex due to different distance patterns. The boundary is composed of four parts, including hyperbola and oval, which constitutes an irregular shape while V*-Diagram can be modeled as a regular shape.

(ii) We extend our methods to construct the bisector of two poly line segments and the corresponding safe region, to support queries over poly lines. Computing the distance from a given point to a poly line segment is more difficult than a line segment. We need to divide the region into several sub-regions to compute the distance. For any point p in one sub-region, the distance from p to a certain poly line is equal to the distance away from the corresponding line segment. Compared to just computing one line segment, it needs more sophisticated geometry inference and theoretical verification.

(iii) Furthermore, in a Cartesian coordinate system, the irregular geometric shape obtained by our ISR is usually represented by a second-degree polynomial. It will be quite time consuming to truly draw such safe region (or compute all the boundaries) and judge whether an object is in the ISR or not. Therefore, we need to explore high-efficient judgment conditions based on the geometric properties of potential boundaries to cut down the side-effect of the irregular shape.

In this paper, we address these challenges and make the following contributions:

– To the best of our knowledge, this is the first tailored technique to handle the line segment and poly line segment oriented MkNN query with no predefined query trajectory. We formulate two types of regions based on V*-Diagram, namely the safe region w.r.t. a line segment and the line segment oriented fixed-rank region, to form an integrated safe region for processing the query.
– We exploit the properties of the line segment oriented integrated safe region to reduce the query processing cost, and obtain a highly efficient algorithm to process the line segment oriented MkNN query.
– We perform the extensive theoretical analysis and comprehensive experimental evaluation on the proposed algorithm. The results show that our algorithm outperforms the baseline algorithm by up to two orders of magnitude.

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 about the V*-diagram and the Line Segment Divided Region. In Section 4, we introduce the LV*-Diagram with line segment oriented fixed-rank region, line segment oriented IRU, integrated safe region. And we also extend the problem to poly line segments. We present the algorithm to process LMkNN query in Section 5. We report the experimental evaluation in Section 6 and conclude the paper in Section 7.

2 Related work

MkNN is evolved from the kNN problem where we want to acquire the k nearest neighbors of a given query point. Many kNN algorithms were proposed based on spatial hierarchical structures, such as R-tree [9], which maintains the distance from the query point to all data objects. And it can be traversed by a depth-first (DF-kNN) [22] or a best-first (BF-kNN) [11] manner to find the kNNs.

For MkNN, the query results must be responded continuously with the movement of a query point. And there are two different research streams. The first are *sampling*-based approaches [23], which are suitable for general scenarios but inefficient in terms of the computation load, I/O cost and communication cost. The reason is that they lack built-in support for the incremental computation for moving objects. On the other hand, *safe region*-based approaches are widely used for MkNN queries, which provides continuous answers and reduces the processing and communication costs. Particularly, the Voronoi Diagram (VD) [13, 20] is a classical solution. The k-version of VD is called kVD which can be used to find the kNNs. However, kVD must pre-compute all kVD cells, which incurs expensive computation and storage costs. Although extensions like TPkNN [24] and CkNN [25] support kNN processing with local information, the query points are limited to given linear trajectories. Furthermore, RIS-kNN [27] computes kVD cells locally based on a spatial index, which avoids the pre-computation. Note that the aforementioned solutions are not adaptive to the variable k value.

As an improved variant, the Incremental Rank Updates (IRU) method [14] can incrementally compute a neighboring nVD cell from the current cell. By leveraging the bisector between two rank-adjacent objects, IRU acquires the new ranking by only swapping the two objects of its bisector, which avoids computing the whole nVD. Nevertheless, the updating cost of IRU is still substantial because it must access global data objects to guarantee that the update is *safe* and *incremental*. To address this issue, Nutanong et al. proposed the V*-Diagram [18] algorithm to obtain an approximate safe region with only a certain number of data objects. Intrinsically, V*-Diagram introduces x auxiliary data objects and computes a fixed-rank region (FRR) including the $(k + x)$NNs of the given query point q, which guarantees the effectiveness of kNNs as long as the query point moves in the safe region. To the best of our knowledge, the efficient approximation of V*-Diagram has been widely used to spatial network problems [19]. However, these techniques only focus on handling data objects modeled as points instead of line segments.

In addition, some studies focus on the moving or continuous nearest neighbor queries in obstructed spaces. Gao et al. [5, 7] discuss the continuous obstructed kNN assuming the query point q moves on a given trajectory. By dividing the trajectory into different safe intervals, the obstructed kNN results can be continuously responded. To deal with obstacles, a novel concept of control points is proposed. Li et al. [17] study the problem of evaluating moving obstructed kNN in a space without any trajectories given. Effective safe-region

based methods are proposed to avoid redundant computation. In the space blocked by obstacles, another category of queries is visible kNN queries. Totally different from obstructed queries, visible kNN queries only consider k nearest data objects which are not blocked by any obstacles instead of calculating obstacle-avoiding distance. In [6] and [8], the continuous visible k nearest neighbor (CVNN) queries are formulated and efficient speed up algorithms for CVNN query processing are proposed, assuming that both data points and obstacles are indexed by R-trees. There are also some studies on MkNN variants. For example, Aly et al. [2] show how queries with two kNN predicates can be processed; Ali et al.[1] explore the probabilistic moving nearest neighbor queries; Hu et al. [12] investigate the MkNN over moving objects. None of these solutions aim at our proposed problem.

In spatial databases, line segments are recognized as fundamental spatial information representations. kNN queries over line segments objects can be processed using R-tree and its variants [3, 9] based on pruning methods. For the snap-shot kNN queries, the technique is efficient and we also leverage such method once the kNN result needs to be updated. When we consider continuous kNN maintenance, extending TPkNN [25] for points to adapt to line segment data may be theoretically feasible, but the trajectory of the moving query point must be assumed to be predefined. In addition, some studies focus on Voronoi diagram analysis and construction over line segment data [10], while computing a k-order Voronoi diagram ($k>1$) for line segment data will incur tremendous costs which makes it not applicable in the real applications especially when we require the kNN result to be order sensitive (i.e., ordered k-order diagram). Besides, there exists the relative scheme [4] for NN query devoting to the case when the query objects are modeled as line segments and the data objects are points while our setting represents the data objects as line segments or poly line segments. For the obstructed or visible MkNN queries [6, 7], obstacles are usually represented by line segments while data objects are modeled as line segments in our work. In conclusion, despite the bulk of NN query related literature, no existing solutions are efficient in our problem because we aim to continuously evaluate order-sensitive k nearest line segment neighbors without any trajectory assumption of the query point.

3 Preliminaries

3.1 V*-diagram

Our basic idea is inspired by V*-Diagram [18], the state-of-the-art method for the MkNN query with no predefined query trajectory in space.The key steps are introduced as follows.

Step 1. Let q_c be the current position of the query object q and z be its $(k+x)^{th}$ nearest data object. Then a known region is computed as a centered at q_c with its radius being $dist\,(q_c, z)$. The safe region w.r.t. a data object p, $S\,(q_c, p, z)$, is computed as a region that, when q stays in the region, p is nearer to q than any data object p' outside the known region. The final equation of $S\,(q_c, p, z)$ is computed as $dist\,(q', p) + dist\,(q_c, q') \leq dist\,(q_c, z)$. which can be proved to satisfy the property $dist\,(q', p) \leq dist\,(q', p')$. According to the definition, the safe region w.r.t. p is essentially an ellipse in the Euclidean space where q_c and p are its two foci and $dist\,(q_c, z)$ is its major axis length. Furthermore, as long as q is inside the intersection of the safe regions w.r.t. the k nearest neighbors of q_c, it can guarantee that any data object p' outside the known region cannot be closer to q than any of those k data objects.

Before introducing step 2, we first explain the concept of fixed-rank region (FRR) and incremental rank updates(IRU) which will be used in step 2. Fixed-rank region (FRR) was introduced in [14]. When the query point is in this region, the ranking of the data objects is fixed based on their distances to the query point. When all the data objects are considered, FRR equals to an ordered kVD cell with k=n where n is the total number of objects. Furthermore, incremental rank updates(IRU) was proposed in [14] as an efficient technique to construct and maintain a new FRR when the ranking of the involved data objects changes. The core idea of FRR is based on the observation that only rank-adjacent objects can swap their ranks, and thus constructing the FRR of n objects requires at most n-1 bisectors of the n-1 pairs of rank-adjacent objects. Specifically, by maintaining a rank-sorted list of objects and its corresponding list of bisectors of pairs of rank-adjacent objects (rank-adjacent bisectors), continuous monitoring of the ranking of the objects can be implemented. Each time a bisector is crossed by the query point, the ranks of the two corresponding objects are swapped and the list of rank-adjacent bisectors are updated based on IRU. In V*-Diagram, the involved data points for a FFR is reduced from n to k+x to improve the efficiency. The correctness of the kNN query result is guaranteed by combining FFR with the concept of safe region w.r.t. a point.

Step 2. The V*-Diagram algorithm further computes FRR where the order of distances of the $(k + x)$ data objects to q does not change, either. Formally, the fixed-rank region of a list L_{k+x} of $(k + x)$ ranked data objects, $\eta\langle p_1, p_2, \ldots, p_{k+x}\rangle$, is defined as the intersection of the bisectors between p_i and p_{i+1}, $H(p_i, p_{i+1})$, where p_i is nearer than p_{i+1} $(i \in [1 \ldots k + x - 1])$:$\eta < p_1, p_2, \ldots, p_{k+x} >= \bigcap_{i=1}^{k+x-1} H(p_i, p_{i+1})$. This region is constructed and maintained with IRU algorithm to keep the $(k + x)$ objects sorted according to their distances to q.

Step 3. The intersection of the safe regions w.r.t. the k data objects and the fixed-rank region is the Integrated Safe Region (ISR), denoted by $\Omega(q_c, L_{k+x})$. Formally, $\Omega(q_c, L_{k+x}) = \eta(L_{k+x}) \cap \left(\bigcap_{i=1}^{k} S(q_c, p_i, z)\right)$, where p_i denotes the i^{th} nearest data object of q.

Figure 1 shows an example where $k = 2$ and $x = 2$. When the query object q is at the location q_c, a 4NN search retrieves the 4 nearest data objects $\eta\langle p_1, p_3, p_2, p_6\rangle$. The

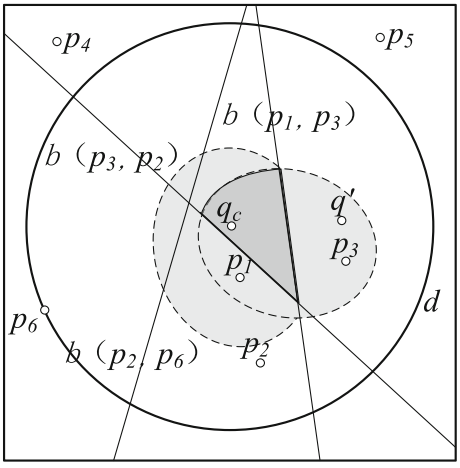


Figure 1 Integrated safe region (k=2,x=2)

ellipses filled with horizontal lines and vertical lines denote $S(q_c, p_1, p_6)$ and $S(q_c, p_3, p_6)$, respectively. Then as long as q remains in the grey region $\eta\langle p_1, p_3, p_2, p_6\rangle \cap S(q_c, p_3, p_6) \cap S(q_c, p_1, p_6)$, the 2NN of q will not change.

3.2 Line segment divided region

Before discussing the details of LV*-Diagram, we first introduce some fundamental concepts for cases where data objects are modeled as line segments. Let q be the query point. The i^{th} line segment (data object) is described by s_i and the corresponding two endpoints e_i^l and e_i^r. Here, we call e_i^l the left endpoint and e_i^r the right endpoint respectively. And the Euclidean distance of two points p_1 and p_2 is formalized as $dist(p_1, p_2)$. Then, the data space is partitioned into three regions by two parallel lines perpendicularly to s_i whose perpendicular foots are e_i^l and e_i^r respectively. Within each region, the distance from q to s_i, can be represented by Formula (1).

$$dist(q, s_i) = min\{dist(q, p) \mid \forall p \in s_i\} \tag{1}$$

Different from point-to-point distance, $dist(q, s_i)$ may vary according to different spatial relationship between q and s_i. An example is shown in Figure 2a. At the beginning, the query point is at q_1, $dist(q, s_i)$ is the distance from q_1 to e_i^l, the nearest endpoint of s_i. When the query point moves from q_1 to q_2, $dist(q, s_i)$ changes to the distance from q_2 to its projection on s_i. Then the query point moves on to q_3, and now $dist(q, s_i)$ is the distance from q_3 to e_i^r. We call the different ways to compute $dist(q, s_i)$ distance patterns.

For the ease of computation in future steps, we first partition the data space into such regions that within each region the distance pattern for any line segment does not change. To do this, we construct two perpendicular lines (the dotted lines in Figure 2a) for s_i with one crossing e_i^l and the other crossing e_i^r. In this way, the data space is partitioned into three parts: (1) one inside region R_i^m, which is bordered by the two perpendicular lines (and containing the line segment) and (2) two outside regions R_i^l and R_i^r, each of which is only bordered by the line crossing e_i^l and e_i^r respectively. The two perpendicular lines are called *line segment divided lines*. Within the inside region, $dist(q, s_i)$ is evaluated as the distance from q to its projection on s_i (we denote q's projection on s_i by $prj(q, s_i)$); otherwise, $dist(q, s_i)$ is evaluated as the distance from q to the corresponding endpoint of s_i. For each

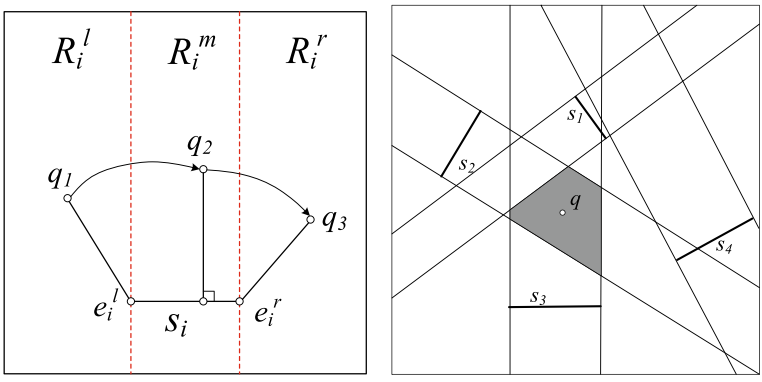


Figure 2 The definition of distance and LDR

line segment, we can do the same partitioning, and the data space is partitioned into a set of *line segment divided regions (LDR)*, which is formulated as follows.

Definition 1 (Line segment divided region, LDR). Given a set of line segments $\{s_1, s_2, \ldots, s_n\}$, an LDR is an element of the set D, where

$$D = \bigcap_{i=1}^{n} R_i,\ with\ R_i \in \{R_i^m, R_i^l, R_i^r\} \tag{2}$$

As an example in Figure 2b, the grey region where the query point q locates is an LDR computed by $R_1^r \bigcap R_2^m \bigcap R_3^m \bigcap R_4^l$. When the query point moves, it could possibly cross different LDRs. Next, we aim to prove the relationship of the distance pattern with line segment divided lines and corresponding LDRs in Lemma 1 and Theorem 1.

Lemma 1 *Let q be the moving query point and s be a certain line segment. If q crosses any line segment divided line of s, the distance pattern from q to s alternates; otherwise it remains unchanged.*

Proof (1) If q is in the inside region of s, $dist(q, s)$ is the distance from q to its projection on s. When q crosses any line segment divided line of s, q is in the outside region of s now. If the distance pattern remains unchanged, the current projection is on the extension line of s. So the distance pattern must alternate. (2) If q is in the outside region of s, $dist(q, s)$ is the distance from q to the nearer endpoint (denoted by e) of s. When q crosses any line segment divided line of s, q is in the inside region of s now. The distance from q to its projection s is smaller than $dist(q, e)$. So the distance pattern alternates. □

The converse proposition of Theorem 1 is also true. The proof is omitted.

Theorem 1 *Let q be the query point in an LDR, the distance pattern from q to any line segment does not change if q remains in the LDR.*

Proof The boundaries of an LDR are made up of the line segment divided lines. According to Lemma 1, as long as q does not cross any line segment divided line (q remains in the LDR), the distance pattern stays unchanged. □

4 The LV*-diagram

The LV*-Diagram proposed in this paper computes an approximated safe region in which q's kNNs (as line segments) do not change when the query point q moves. LV*-Diagram involves two basic steps to compute the approximation: (1) compute a fixed-rank[1] region (FRR) given the $(k + x)$NNs of q. Within the FRR, the ranking[2] of the $(k + x)$NNs does not change. (2) for each of the kNNs of q, compute a safe region with regard to the line

[1]Fixed-rank means the rank of an object doesn't change. It is specifically decorate a region (i.e. fixed-ranked region) in our paper, which indicates the ranks of all the objects in such region keep fixed.

[2]Ranking is used to represent the sequence with the objects sorted in this paper. For example, we can say the ranking (of s_1, s_2 and s_3) is $\langle s_1, s_3, s_2\rangle$.

segment s_i. Extending this framework to accommodate line segments and poly line segments requires targeted improvements and complicated geometrical analysis. In general, the construction models of line segment oriented FRR and the safe regions with regard to line segments are unfolded in Sections 4.1 and 4.2 respectively.

4.1 Line segment oriented FRR & line segment oriented IRU

A line segment oriented fixed-rank region (FRR) is such a region that tries to keep the $(k+x)$ objects sorted according to their distances to the query point. Essentially, the line segment oriented FRR is the closed region obtained by intersecting the $n-1$ bisectors of any two adjacent objects in the rank[3] list, where n is the number of data objects in the rank list. The line segment oriented FRR can be incrementally updated by extending the Incremental Rank Updates (IRU) technique to adapt to the line segment objects. Specifically, when the query point exits the current line segment oriented FRR, it must have crossed some bisector; then we know that *only* the ranking of the two data objects corresponding to the crossed bisector needs to change. The new safe region is computed by intersecting the new bisectors for the new rank list. It is non-trivial to implement this for line segments objects.

As discussed in Section 3, the whole data space is partitioned into multiple LDRs. Computing the bisector of any two objects must be distinguished according to the distance pattern of each LDR. We denote the bisector of two data objects a and b (regardless of points or line segments) by $bisect(a, b)$. Figure 3a gives an example of the bisector of two line segment data objects. The construction is given in Table 1.

Finally, $bisect(s_1, s_2)$ is constructed as an irregular line by connecting all the above curves. Furthermore, the line segment oriented FRR is obtained by intersecting all the bisectors of any two rank-adjacent line segment objects. An example is depicted in Figure 3b. The FRR with regard to line segment data objects is formally defined as follows.

Definition 2 (Fixed-Rank Region). Given an ordered list L of line segments $\langle s_1, s_2, \ldots, s_n\rangle$, the line segment oriented fixed-rank region (FRR) of L is:

$$F\langle s_1, s_2, \ldots, s_n\rangle = \bigcap_{i=1}^{n-1} H_{s_i s_{i+1}} \tag{3}$$

$H_{s_i s_{i+1}}$ is the region where for an arbitrary point p in Euclidean space, $dist(p, s_i) \leqslant dist(p, s_{i+1})$. $F\langle s_1, s_2, \ldots, s_n\rangle$ can be abbreviated as $F\langle L\rangle$.

To maintain the line segment oriented FRR with a given k value, we leverage the line segment oriented IRU algorithm. The line segment oriented IRU establishes two lists: the *rank list* and the *rank-adjacent bisector list* based on the distance from the query point to the line segments. Once the moving query point crosses a bisector, the ranks of the two corresponding segments are swapped and the list of rank-adjacent bisectors is updated, and thus a new line segment oriented FRR with the new ranking is obtained.

An example is given in Figure 3b and c. The grey region is $F(L)$. Suppose a query point q moves from q_1 to q_2. In Figure 3b, q is at q_1 and the ranking is $\langle s_1, s_3, s_2, s_4\rangle$, and the corresponding list of bisectors is $\langle B_{13}, B_{23}, B_{24}\rangle$. In this region, B_{13} and B_{24} are parabolas whereas B_{23} is an angle bisector. In Figure 3c, q moves from q_1 to q_2 by crossing B_{24},

[3]The rank of an object means the object's position in a list of objects sorted by their distances to some other objects.

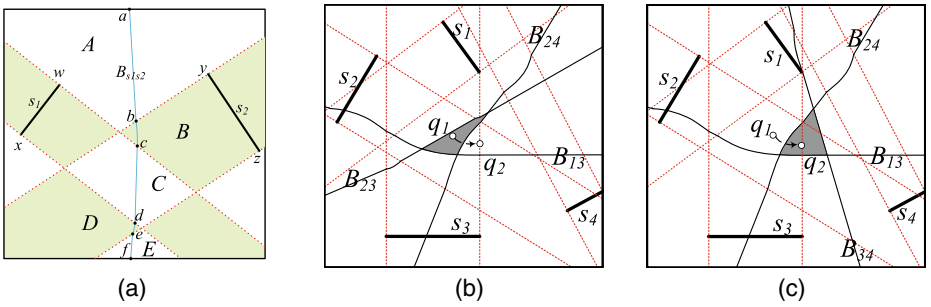


Figure 3 Bisectors and IRU for line segments

which causes the ranks of s_2 and s_4 to swap. The new ranking is $\langle s_1, s_3, s_4, s_2 \rangle$. The new line segment oriented FRR constrained by $\langle B_{13}, B_{34}, B_{24} \rangle$ is depicted as the grey region in Figure 3c.

Note that an line segment oriented FRR may span across multiple LDRs. Even if the query point may be in different LDRs, the ranking of the data objects does not change as long as the query point is still in the line segment oriented FRR. But in the implementation, calculation of the distance from the query point to any line segment object must conform to the corresponding LDR.

4.2 Safe region with regard to a line segment object

The line segment oriented FRR computed in the last section guarantees that the order of the $(k + x)$ objects in the ranking does not change. But the line segment oriented FRR cannot exclude such situations that some object other than the $(k + x)$ objects could become one of the kNNs. For example, the original ranking is $\langle a, b, c \rangle$ (with $k = 2$ and $x = 1$). When the query point moves in $F\langle a, b, c \rangle$, the order of a, b and c is enforced, but another unknown object d could get in and the actual ranking becomes $\langle a, d, b, c \rangle$. This possibility cannot be identified by the line segment oriented IRU approach, since d is not explicitly maintained in the rank list. To make sure, for example, a and b are still the 2NNs, we must further constrain the safe region to exclude such possibilities that objects like d could replace some object and get in the kNNs.

As extended from V*-Diagram [18], the LV*-Diagram introduces the notion of a *safe region with regard to a line segment* (LSR). The LSR of a line segment s_m guarantees that if the query point q stays in this region, for any data object s_n other than the $(k + x)$NNs, $dist(q, s_m) \leq dist(q, s_n)$ always holds. We will then use an example to show how to construct the LSR.

Table 1 Bisectors in different LDRs

LDR	Equivalence	Bisector Type	Bisector
A	$dist(q, s_1) = dist(q, w), dist(q, s_2) = dist(q, y)$	Perpendicular bisector	ab
B	$dist(q, s_1) = dist(q, w)$	Parabola	bc
C	$dist(q, s_1) = dist(q, s_2)$	Angle bisector	cd
D	$dist(q, s_1) = dist(q, x)$	Parabola	de
E	$dist(q, s_1) = dist(q, x), dist(q, s_2) = dist(q, z)$	Perpendicular bisector	ef

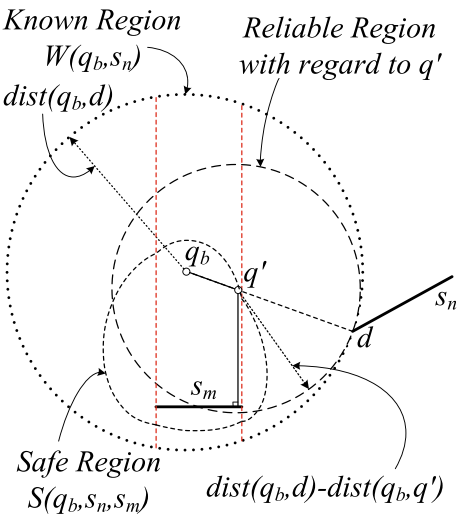


Figure 4 The safe region with regard to a line segment

In Figure 4, let q_b be the position of q where the last BF-kNN is called (i.e., the $(k+x)$NNs have been determined). Let line segment s be the $(k+x)^{th}$NN of q_b and line segment s_n be one of the kNNs of q_b. We construct a *Known Region*, denoted by $W(q_b, s_n)$, as a disk centered at q_b with the radius $dist(q_b, s_n)$. So our target is to construct a region S, as long as q' stays in S, we can get $dist(q', s_m) \leqslant dist(q', s_n)$, where s'_n can be any line segment not in the $(k+x)$NNs of q_b. According to the definition of $KnownRegion$, we can get for any s'_n not in the $(k+x)$NNs of q_b, $dist(q_b, s_m) < dsit(q_b, s'_n)$. So if we can construct a region for q' to satisfy $dist(q', s_m) \leqslant dist(q_b, s_n) - dsit(q_b, q')$, we can get $dist(q', s_m) \leqslant dist(q_b, s'_n) - dsit(q_b, q')$. Furthermore, according to the triangle inequality, $dist(q_b, s'_n) < dist(q', s'_n) + dsit(q_b, q')$, so we can get $dist(q', s_m) \leqslant dist(q', s'_n) + dsit(q_b, q') - dist(q_b, q') = dist(q', s'_n)$. That means the constructed region can satisfy the desired property of LSR.

Definition 3 (Safe Region with regard to a line segment). Given a known region $W(q_b, s_n)$ and a line segment s_m within $W(q_b, s_n)$, the safe region with regard to s_m is:

$$S(q_b, s_n, s_m) = \{q' : dist(q', s_m) + dist(q_b, q') \leqslant dist(q_b, s_n)\} \tag{4}$$

The safe region with regard to an object in the original V*-Diagram is actually an oval. But for the case of LSR, the safe region is much more complex due to different distance patterns. The boundary of an LSR is composed of four parts. Each of the two parts in the *outside regions* of s_m is a chunk of a corresponding oval, since in such cases, $dist(q, s_m)$ is the distance from q to one of the endpoints of s_m. Each of the two parts in the *inside region* of s_m is the chunk of such a trajectory that the sum of the distance to a fixed point (q_b) and a fixed line (corresponding to s_m) is a constant. We will prove that for the latter case, the trajectory is actually a parabola.

Theorem 2 *For 2D Euclidean space, each of the two segments of the boundary of an LSR in the inside region is a part of a parabola.*

Proof This problem is equivalent to a locus problem of points of which the sum of distances to a fixed point and a fixed line is a constant. Suppose F is the fixed point, l is the fixed line, and the distance from F to l is $k(k \geqslant 0)$. We establish a rectangular coordinate with the y-axis coincided with l and the x-axis crossing F (shown as the upper part of Figure 5). The coordinate of F is $(k, 0)$. Assume an arbitrary point $M(x, y)$ and the distance from M to l is d. We are to prove equation $\mid MF \mid +d = t$ holds, where t is a constant ($dist(q_b, d)$ in Figure 4). Apparently, if $t < k$, the locus of point M does not exist. If $t \geqslant k$, from $\mid MF \mid +d = t$ we have

$$\sqrt{(x-k)^2 + y^2} + \mid x \mid = t \tag{5}$$

(1) If $x \geqslant 0$, equation (5) $\Rightarrow \sqrt{(x-k)^2+y^2} = t - x \Rightarrow (x-k)^2 + y^2 = (t-x)^2$, $(t - x \geqslant 0) \Rightarrow y^2 = 2(k-t)(x - \frac{k+t}{2})(0 \leqslant x \leqslant t)$.
If $t > k$, the locus of point M is a section of parabola p_1 whose vertex is at $D(\frac{k+t}{2}, 0)$ and left open; if $t = k$, the results corresponds to line segment OF.

(2) If $x < 0$, the proof is similar to (1), and the corresponding parabola is p_2 whose vertex is at $C(\frac{k-t}{2}, 0)$ and right open.

□

The final LSR is the closed region by intersecting the above ovals and parabolas. For each line segment in the kNNs, we can compute a corresponding LSR. If the query point exits the LSR of any line segment, a new ranking for the $(k+x)$NNs should be re-evaluated with the BF-kNN algorithm, which incurs computation overhead and communication with the server.

4.3 Safe region with regard to a polyline segment object

Intrinsically, some real-world objects (i.e., roads) may be better modeled as polyline segments which consist of multiple consecutive line segments. In this section, we extend our methods proposed in Sections 4.1 and 4.2 in terms of constructing the bisector of two polyline segments and the corresponding safe region, to support queries over polyline segments.

Before discussing the issue of bisectors, we first describe how to compute the distance from a given point to a polyline segment. As illustrated in Figure 6a, for any polyline

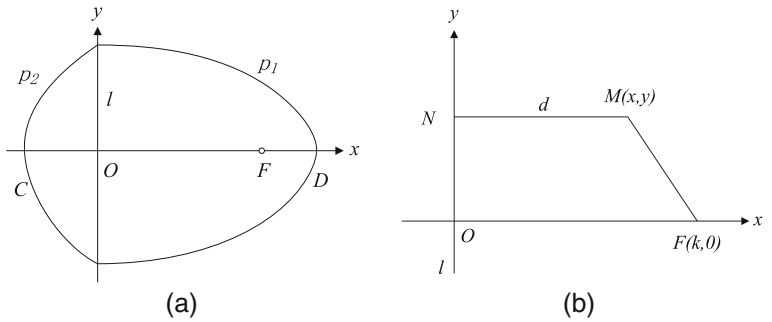


Figure 5 The geometrical characteristic of an LSR's boundary in the inside region of a segment

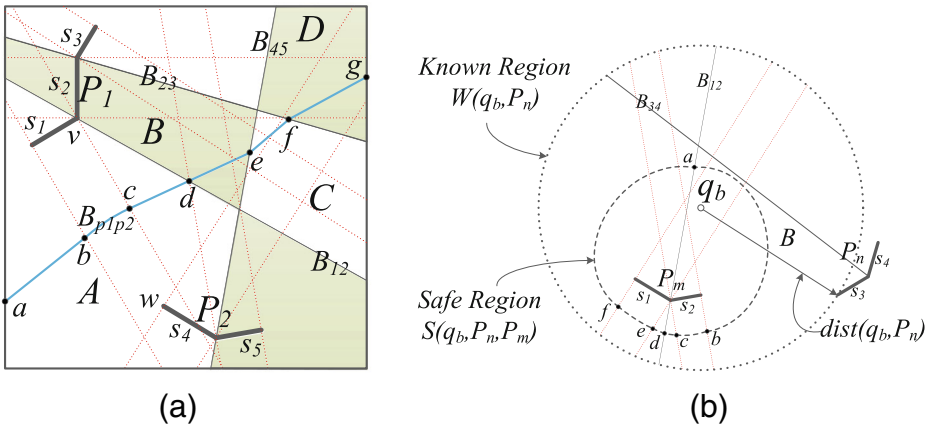


Figure 6 The safe region with regard to a polyline segment

segment, we build the angel bisector of its two consecutive line segments. For example, the polyline segment P_1 constructed by line segments s_1, s_2 and s_3, has two angel bisectors B_{12} and B_{23}, which divide the data space into 3 regions. Note that all angle bisectors of polyline segments divide the region into several sub-regions, termed *PDRs*. For any point p in one PDR, the distance from p to a certain polyline is equal to the distance away from the corresponding line segment.

The bisector of two polylines segments can be constructed by building the bisector of its every line segment. Figure 6a shows an example of constructing the bisector between P_1 and P_2. Here, PDRs are distinguished by shadows. In each PDR, we first build the bisector between two line segments by the line segment oriented IRU method, and then integrate these bisectors to compute the bisector between P_1 and P_2. The details of this process are given in Table 2.

After that, we can construct the safe region for a polyline by decomposing it into multiple line segments. As demonstrated in Figure 6b, suppose that q_b is the location of the query point q after invoking the BF-kNN algorithm. While, P_n and P_m denote the (k+x) nearest neighbor and some neighbor whose distance away from q is smaller than that of P_n. First, for common segments of P_m and P_n, we build their bisectors, B_{12} and B_{34}. Obviously, $dist(q_b, P_n) = dist(q_b, s_3)$. Then, the known area $W(q_b, P_n)$ can be obtained based on $dist(q_b, s_3)$. Finally, since P_m consists of s_1 and s_2 whose bisector B_{12} divides the area into A and B. In A, the safe region of s_1 is surrounded by $\widehat{af}$, $\widehat{fe}$ and $\widehat{ed}$. Especially, $\widehat{fe}$ is the inner segment safe region, and $\widehat{af}$ and $\widehat{ed}$ are outer segment safe regions. For B, the safe region of s_2 consists of $\widehat{ab}$, $\widehat{bc}$ and $\widehat{cd}$. Similarly, $\widehat{bc}$ is the inner segment safe region

Table 2 Bisectors in different PDRs

PDR	Equivalence	Bisector
A	$dist(q, P_1) = dist(q, s_1), dist(q, P_2) = dist(q, s_4)$	ad
B	$dist(q, P_1) = dist(q, s_2), dist(q, P_2) = dist(q, s_4)$	de
C	$dist(q, P_1) = dist(q, s_2), dist(q, P_2) = dist(q, s_5)$	ef
D	$dist(q, P_1) = dist(q, s_3), dist(q, P_2) = dist(q, s_5)$	fg

and $\widehat{ab}$ and $\widehat{cd}$ are outer segment safe regions. Note that line segment oriented *FRR* can be constructed for polyline segment objects with the same techniques introduced in Section 4.1.

4.4 Integrated safe region

Now we can obtain the final safe region by intersecting the current line segment oriented FRR (obtained in Section 4.1) of the $(k + x)$ data objects and the LSRs with regard to the k nearest neighbors (obtained in Section 4.2). We call the final safe region the *integrated safe region* (ISR). The formal definition of ISR is given as follows, and we can prove that the ISR is the region that satisfies the requirement for a safe region enforcing the same kNNs for the query point. In addition, based on Sections 4.1 and 4.3. the integrated safe regions for polyline segment objects can be constructed in the same way. We omit the details in this section and Section 5.

Lemma 2 $F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j) \cap S(q_b, s_n, p_i) = F\langle s_i, s_j\rangle \cap S(q_b, s_n, p_j)$

Proof For any point $p \in F\langle s_i, s_j\rangle$, p satisfies

$$dist(p, s_i) \leqslant dist(p, s_j). \tag{6}$$

For any point $p \in S(q_b, s_n, s_j)$, by definition p satisfies

$$dist(p, s_j) + dist(q_b, p) \leqslant dist(q_b, s_n). \tag{7}$$

For any point $p \in F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j)$, by adding (6) and (7),

$$dist(p, s_i) + dist(q_b, p) \leqslant dist(q_b, s_n). \tag{8}$$

Inequality (8) shows that giving a point p which is in $F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j)$, we can get $p \in S(q_b, s_n, s_i)$. That is $F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j) \subseteq S(q_b, s_n, s_i)$, i.e. $F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j) \cap S(q_b, s_n, s_i) = F\langle s_i, s_j\rangle \cap S(q_b, s_n, s_j)$. □

Theorem 3 *For* $k \geqslant 2$, $F\langle s_1, s_2, \ldots, s_k\rangle \cap \left(\bigcap_{i=1}^{k} S(q_b, s_n, s_i)\right) = F\langle s_1, s_2, \ldots, s_k\rangle \cap S(q_b, s_n, s_k))$.

The theorem can be proved by induction using Lemma 2. We omit the details here due to limited space.

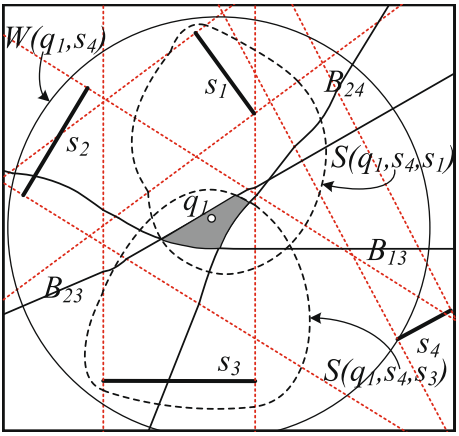
Definition 4 (Integrated Safe Region). Given a $(k+x)$NN list of q_b and L, s_n is the farthest retrieved line segment to q_b. The integrated safe region with regard to q_b and L is:

$$I(q_b, s_n, s_i, L) = F(L) \cap S(q_b, s_n, s_k)) \tag{9}$$

Theorem 4 *The kNN query results for any point in the ISR is the same, that is the ISR is a reliable safe region.*

Proof (1) The ranking of the $(k + x)$ line segments is fixed in region $F(L)$, so no data object in the $((k + 1) \sim (k + x))$NN can become one of the kNNs. (2) The result of kNNs is fixed in $S(q_b, s_n, s_i)$, so no data object other than the $(k + x)$NNs can become one of the kNNs. To sum up, for any point in ISR, the k nearest neighbors are unchanged. □

Figure 7 An example of ISR



An example is given in Figure 7. There are four line segments and q_1 is the current position of the query point q. The ranking of objects at q is $\langle s_1, s_3, s_2, s_4\rangle$. Suppose k=2 and x=2. So the final safe region is the intersection of $F(\langle s_1, s_3, s_2, s_4\rangle)$ and $S(q_1, s_4, s_3)$.

5 Algorithms

We now present the algorithms to process line segment oriented MkNN queries, which implement the proposed LV*-Diagram. First, the procedure of constructing ISR is illustrated in Algorithm 1.

Algorithm 1: Line segment oriented Integrated Safe Region Construction (ISRC)

Input : the query point q_b, the k value, the x value
Output: the integrated safe region ISR
1 $L \leftarrow$BF-kNN$(q_b, k + x)$
2 $D \leftarrow$ Create line segment divided region of L
3 $FRR \leftarrow$ Create fixed-rank region of L
4 $s_n \leftarrow L$.Item$(k + x)$
5 $s_k \leftarrow L$.Item(k)
6 $S_k \leftarrow S(q_b, s_n, s_k)$ /* S_k: the safe region with regard to s_k. */
7 $ISR \leftarrow S_k \cap FRR$
8 **return** (L, S_k, D, FRR, ISR)

Based on Algorithm 1, we furthermore present Algorithm 2 to handle the MkNN problems. Especially, when the query point q crosses the border of line segment oriented FRR (line 6 $\sim$ 10), the ranks of the corresponding rank-adjacent objects will change. Then the new line segment oriented FRR is re-calculated using the line segment oriented IRU algorithm (line 7). Note that if the k_{th}NN changes, the new S_k needs to be reconstructed (line 8). After that, ISR is updated (line 10). Otherwise, if q crosses the border of S_k, a new ISR must be computed by Algorithm 1 because the change of the $(k + x)$NNs is unpredictable.

Algorithm 2: Line-segment-oriented MkNN (LMkNN) processing

Input : the new query point q_b and the old query point q_0
Output: the new query result and the safe region when query point is at q_b

```
(L, S_k, D, FRR, ISR) ← ISRC
ReportResult(L.Head(k))
if (the border of S_k is crossed) then
    (L, S_k, D, FRR, ISR) ← ISRC
else
    if (FRR is crossed) then
        Update L and FRR
    if (the k^th NN changed) then
        Update the S_k
    Update the ISR
```

An example to explain our algorithm is given in Figure 8. The initial location of the query point q is at q_1 in Figure 7, and now the 4NNs is $\langle s_1, s_3, s_2, s_4\rangle$ and the ISR is $F\langle s_1, s_3, s_2, s_4\rangle \cap S(q_1, s_4, s_3)$, given $k = 2$ and $x = 2$. In Figure 8b, q crosses B_{24} at γ_1. The ranks of s_2 and s_4 are swapped, and the list L becomes $\langle s_1, s_3, s_4, s_2\rangle$, which causes the $F\langle L\rangle$ and ISR to change. The ISR becomes $F\langle s_1, s_3, s_4, s_2\rangle \cap S(q_1, s_4, s_3)$. In Figure 8b, q crosses $S(q_1, s_4, s_3)$ at γ_2. Algorithm 1 is called to re-calculate the safe region. The new $(k+x)$NNs are $\langle s_1, s_3, s_4, s_2\rangle$ and the corresponding ISR is $F\langle s_1, s_3, s_4, s_2\rangle \cap S(\gamma_2, s_2, s_3)$.

The performance of line segment MkNN is now evaluated supposing that the data objects are uniformly distributed. According to the former analysis in Section 5, when a query point crosses the boundary of S_k, BF-kNN will be triggered. So the frequency of calling BF-kNN, f, is inversely proportional to the length of the trajectory from a query point to the boundary of the safe region. In the worst case, q moves in a straight line, and f is inversely proportional to $dist(q_b, q_e)$, where q_b is the original location of q and q_e is the next location of q. As shown in Figure 4, f is inverse to $dist(q_b, q')$, where $dist(q_b, q') = dist(q_b, d) - dist(q', d)$. According to [18, 26], the distance from q_b to the k^{th}NN is

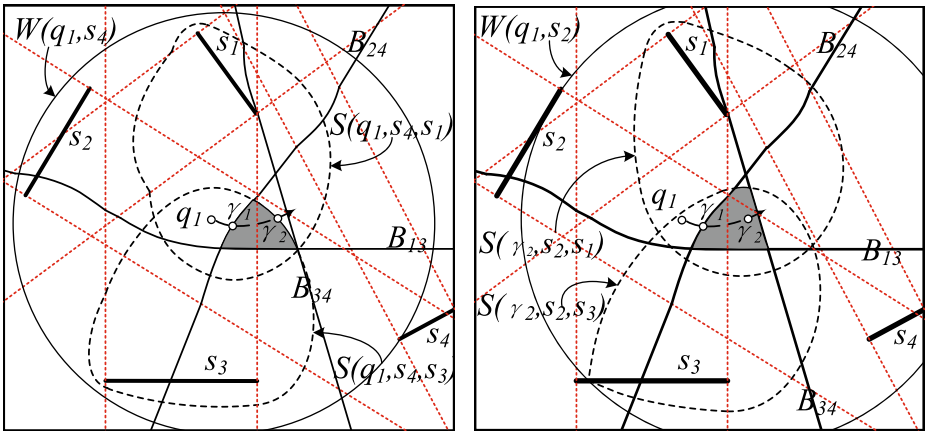


Figure 8 An example of Algorithm 2

$D_k \approx \frac{2}{C_V}\left(1-\sqrt{1-(\frac{k}{N})^{\frac{1}{d}}}\right)$, where $C_V = \frac{\sqrt{\pi}}{[\Gamma(d/2+1)]^{\frac{1}{d}}}$, $\Gamma(x+1) = x \cdot \Gamma(x)$, $\Gamma(x) = 1$. For the two-dimension case, i.e., d=2, $C_V = \sqrt{\pi}$. So $dist(q_b, q')$ can be represented as $D_K \approx \frac{2}{\sqrt{\pi}}\left(1-\sqrt{1-\sqrt{\frac{k}{N}}}\right) - \frac{2}{\sqrt{\pi}}\left(1-\sqrt{1-\sqrt{\frac{k+x}{N}}}\right)$. Because f is the inverse function of $dist(q_b, q_e)$. So, $O(f) = O\left(1/\left(\sqrt{1-\sqrt{k/n}}-\sqrt{1-\sqrt{(k+x)/n}}\right)\right)$. We simplify $O(f)$ as $\frac{1}{\sqrt{1-\sqrt{k/n}}-\sqrt{1-\sqrt{(k+x)/n}}} = \frac{\sqrt{1-\sqrt{k/n}}+\sqrt{1-\sqrt{(k+x)/n}}}{\sqrt{(k+x)/n}-\sqrt{k/n}} \leq \frac{2\sqrt{1-\sqrt{k/n}}}{\sqrt{(k+x)/n}-\sqrt{k/n}} \leq \frac{2}{\sqrt{(k+x)/n}-\sqrt{k/n}} = 2\frac{\sqrt{(k+x)/n}+\sqrt{k/n}}{(k+x)/n-k/n} = 2\frac{\sqrt{(k+x)/n}+\sqrt{k/n}}{x/n} \leq 4\frac{\sqrt{(k+x)/n}}{x/n}$. So, $O(f)$ is $O\left(\frac{\sqrt{(k+x)/n}}{x/n}\right)$. Generally speaking, the value of x and k are comparable. Thus, $(k+x)$ is much smaller than xk. So, $O(f)$ equals $O\left(\sqrt{\frac{kn}{x}}\right)$.

Communication cost When the client side needs to ask the new kNN result, it will request the kNN set from the server. So, the communication cost is incurred when the server returns new result sets. The update frequency of LMkNN is $O\left(\sqrt{\frac{kn}{x}}\right)$ and the number of the result is $k+x$. So, the Communication cost is $O\left((k+x)*\sqrt{\frac{kn}{x}}\right)$.

I/O cost Every time the kNN set is to be updated, the server needs to compute the new result set, which consists of the top $(k+x)$NNs. Therefore, based on R-tree structure, $O(\log n + k + x)$ time is needed on average for each BF-kNN searching. So, the whole I/O cost measured by the data object access is $\left((\log n + k + x)*\sqrt{\frac{kn}{x}}\right)$. Note that if we measure the I/O costs with disk page, because the localities between consecutive kNN searching exist, fewer I/O costs in term of page access number will be incurred.

CPU cost We analyze the CPU cost which is incurred when validating kNN sets in the client side including judging whether the query point is in the safe-region or the line segment oriented FRR border is crossed. So, our method should maintain the LSR with $O(1)$ time and scan a list of line segments sequentially, which requires $O(k+x)$ time to maintain the fixed rank region.

Space cost For FRR, we use $O(k+x)$ space to maintain the rank list and storage the result set. For LSR, we can use the definition of LSR to quickly judge whether an object is in it to avoid storing the real boundaries. So, the space cost is $O(k+x)$.

One alternative method of our problem is to utilize V*-Diagram based on the combination of three key points, i.e., the leftmost point l, center c, and rightmost point r of a line segment s. For each kNN updates, the shortest distance of $dist(q,l)$, $dist(q,c)$, $dist(q,r)$ can be chosen as the approximation of $dist(q,s)$. However, because the update of kNN is only executed when the object leaves the previous ISR, and the shortest distance of $dist(q,l)$, $dist(q,c)$, $dist(q,r)$ will be frequently changed during two consecutive kNN updates when an object is in the ISR constructed by V*-Diagram. Therefore, it is almost as ineffective as just utilizing V*-Diagram based on the center c of a line segment s. Next, we analyze V*-Diagram (which uses $dist(q,c)$ to approximate $dist(q,s)$) and LV*-Diagram in term of efficiency and effectiveness theoretically and experimentally.

Assuming the shortest distance d from a point q to a line segment s should be measured based on q's projection on s, which may have a quite large derivation from the distance d' from q to the center point of s. Theoretically, $0 < d/d' = \sin\alpha < 1$, where $0° < \alpha < 90°$. Therefore, the derivation has no upper bound guaranteed. Obviously, the nearer q is from s or the longer s is (which is quite common in our scenario considering we conduct nearest neighbor queries), the larger the derivation is. Second, because our kNN query result is order sensitive. For two line segments s_1 and s_2, $dist\,(q, s_1) < dist\,(q, s_2)$ cannot infer $dist\,(q, c_1) < dist\,(q, c_2)$ according to the pythagorean theorem and hence it has a large possibility to return wrong results. Similarly, a large derivation can be inferred for two other $dist\,(q, s)$ evaluation patterns. In term of efficiency, the extra costs of our accurate methods are mainly two fold. First, computing d only incur very little extra CPU cost compared with computing d'. Second, because $dist\,(q, s) < dist\,(q, c)$, ISR of LV*-Diagram is a little smaller than that of V*-Diagram which may incur a little more I/O and communication costs. We conduct experimental evaluation in Section 6 to compare these two methods.

6 Experimental evaluation and analysis

6.1 Experiment settings

In this section, the performance of LMkNN is evaluated by comparing with the baseline *sampling*-based method, since LMkNN is the first approach to deal with MkNN queries over line segments. Two real datasets[4] are used in our experiments. One contains the data of 30,674 line segments bounded by MBR in the Germany Road Network and the other records 17,790 line segments bounded by MBR of the Germany Utility Network. Both line segments and polyline segments are contained in the data sets. Two different types of query trajectories are generated: random (R) and directional (D). Each trajectory consists of a series of sampled locations of moving objects. The distance of consecutive locations are controlled by the step length factor which reflects the velocity of objects. We generate 20 different trajectories for each experiment and the average results are obtained. The cumulative total CPU time, I/O cost (in terms of the number of page accesses) and communication cost (in terms of the communication number) for all the sampled locations of each trajectory are recorded. All experiments were implemented in Java and were run on a 3.40GHz Inter Core i7-2600 CPU, 8.00GB RAM and 64 bits windows OS. In each set of experiments, we only alter one parameter and keep others fixed. The major default parameter values are given in Table 3.

6.2 Performance evaluation

The effect of x's value As shown in Figure 9a, we can observe that the CPU time decreases with the increase of x. This is because when x keeps increasing, S_k becomes larger, and hence the frequency to incur a new ISR construction(ISRC) algorithm is reduced. Figure 9b depicts the effects of x on I/O costs measured by the number of page accesses. By analysis, the page access is only incurred by the BF-kNN execution during ISRC. Because a larger x will lead to a larger scale of S_k, and thus the frequency of performing BF-kNN is reduced. On the other hand, the increase of x will cause more points accessed for each

[4] http://www.chorochronos.org/?q=node/54

Table 3 default values of the parameters in the experiment

Parameters	Default value
x	20
k	10
buffer pages	16
number of moving object locations	1000
step length factor	3
line segment number for Gremany Roads	30674
line segment number for Germany Utilitys	17790

BF-kNN execution which will offset the gains of fewer execution times. This is why the overall trend of I/O costs keeps steady with the variation of x. In general, the optimal x value is sensitive to the size and the distribution of datasets. Furthermore, for the Client/Server mode, the server is responsible for receiving the request, querying the k nearest neighbor and returning the results to clients. The client is responsible for computing and updating FRR and LSR while sending the request to the server. The involved communication costs are tested in Figure 9c. The communication will only be incurred when an object crosses the border of ISR and a new ISR needs to be returned. Similarly, a smaller ISR updating frequency is generated with the increase of x, which will dominate the overall communication costs. Here, x is a key parameter to affect the access frequency of the server. Obviously, the sampling methods will not be affected by x in whatever metrics. We can see our algorithm obtains at least an order of magnitude gains for both two different types of query trajectories, i.e., random (R) and directional (D). Particularly, compared to random (R), because directional (D) has a larger possibilities to cross the border of ISR, more costs in term of CPU, I/O and communication will be consequently consumed. Also, Figure 10 conducted on Germany Utility Datasets demonstrates the similar trends. But because fewer line segments are contained in Germany Utility Datasets, the overall execution costs are correspondingly reduced compared to Germany Roads Datasets.

The effect of k In Figures 11 and 12, the effects of k are tested. The augment of k will increase the frequency of ISR construction and the unit construction costs, which will consume more CPU time as a consequence. Compared with *Sampling*, LMkNN obtains at least an order of magnitude gains from the CPU time perspective since the safe region of LMkNN significantly reduces the frequency of performing BF-kNN. Figures 11b and 12b demonstrates that the number of page accesses increases with the augment of k. Because there are two reasons that influence the performance. One is the size of the ISR, another is more

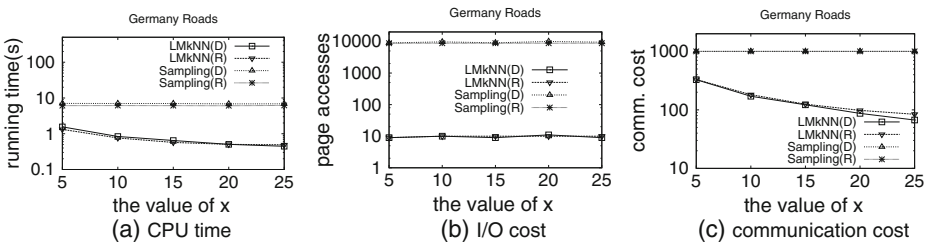


Figure 9 Analyzing the effect of x (Germany Roads)

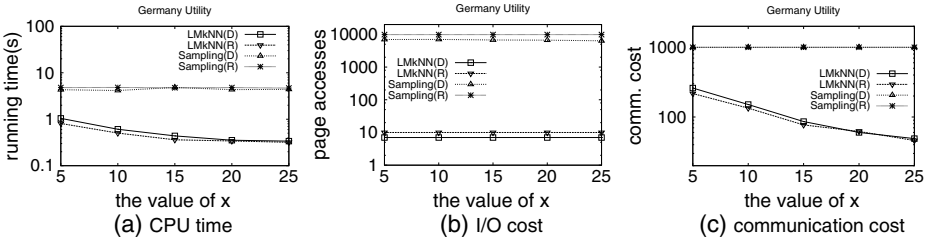


Figure 10 Analyzing the effect of x (Germany Utility)

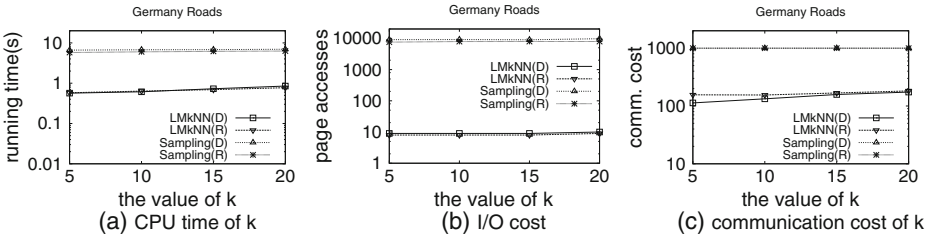


Figure 11 Analyzing the effect of k (Germany Roads)

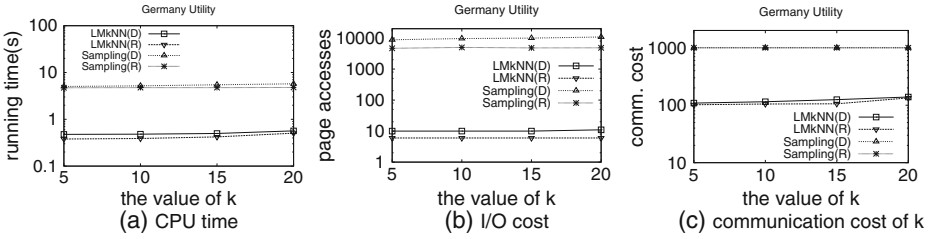


Figure 12 Analyzing the effect of k (Germany Utility)

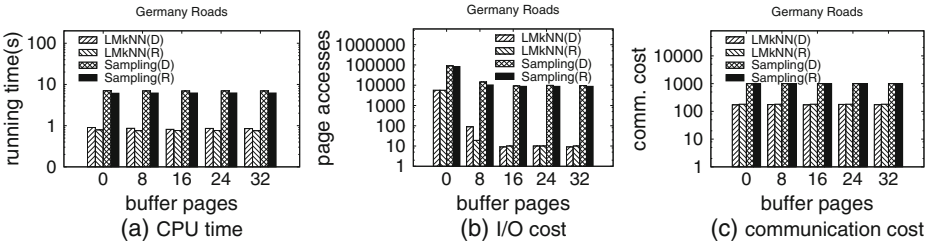


Figure 13 Analyzing the effect of *buffer* (Germany Roads)

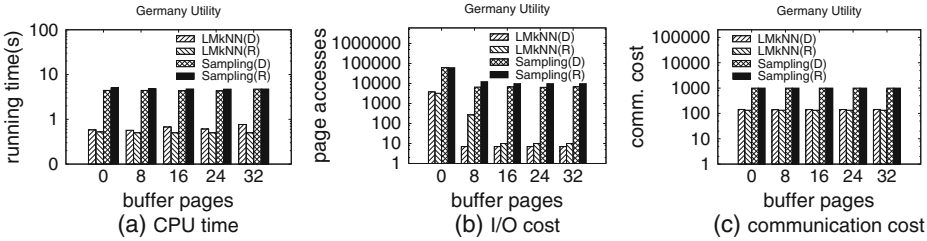


Figure 14 Analyzing the effect of *buffer* (Germany Utility)

data localities. A larger k will increase the size of S_k and thus the frequency of BF-kNN. At the same time, the number of accessed data objects for each BF-kNN calling is also increased for a larger k. However, when we measure the I/O costs with page access, with the shrinking of S_k and more frequency of BF-kNN, the consecutive kNN results have a larger possibility to maintain spatial proximity (i.e., more data localities can be conserved between consecutive BF-kNN extractions) which incurs no extra disk page access for newly introduced BF-kNN operations in most cases. This is why in our tests, when k reaches a large value in the real applications (i.e., 20), the I/O costs still don't display obvious increase trend and generally keep steady. But when k keeps increasing, the high frequency of BF-kNN will be the dominating factor and leads to an increasing trend of I/O costs in the long run. Figures 11c and 12c demonstrate the communication cost for LMkNN and *Sampling*. Also, a larger k will lead to more ISR constructions, and hence results in the increase of communication costs. Similarly, benefitting from the safe region, LMkNN overwhelmingly outperforms *Sampling*.

The effect of *buffer* As shown in Figures 13 and 14, we test the effect of the buffer size on the performance. From the figures, we can conclude the CPU time is not sensitive to the *buffer* size as well as the communication cost. This is because the buffer size has no impact on the number of BF-kNN calling and new ISR construction. For the I/O costs illustrated in Figures 13b and 14b, we can observe the page access is cut down with the increase of the buffer size, which proves that a larger buffer size can contribute to leverage the data locality during the consecutive BF-kNN execution.

The effect of the number of moving object locations Figures 15 and 16 describe the effect of the number of sampled moving object locations. In this set of experiment, we vary the number of sampled locations in the query trajectory from 200 to 1,000. Obviously, for each new location, a new kNN search will be executed by sampling-based methods. And

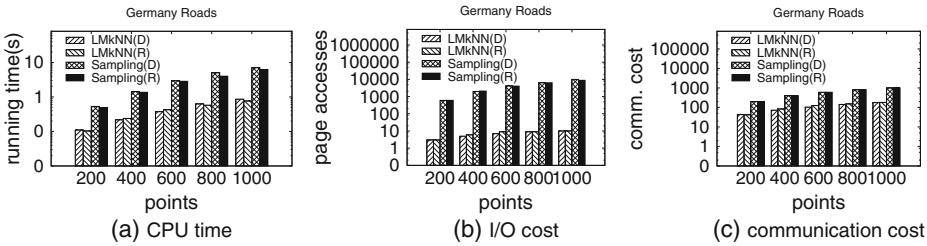


Figure 15 Analyzing the effect of the number of location updates (Germany Roads)

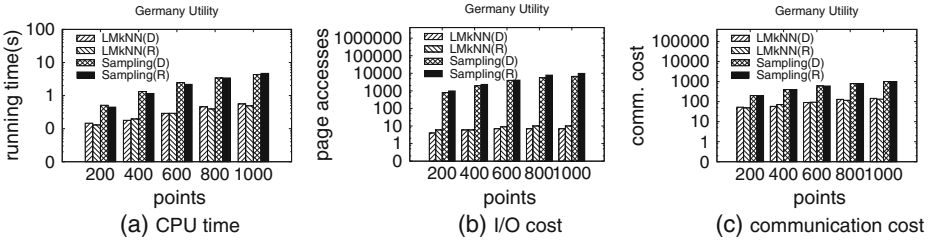


Figure 16 Analyzing the effect of the number of location updates (Germany Utility)

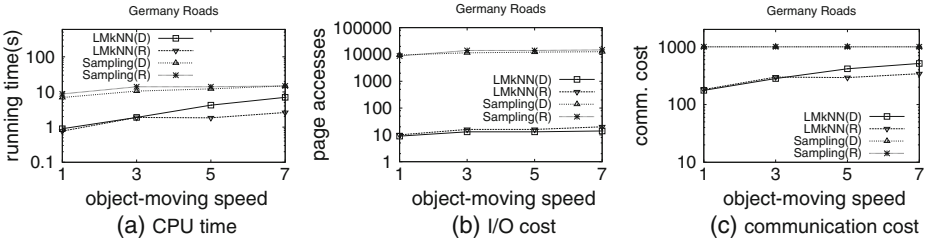


Figure 17 Analyzing the effect of the point-moving speed (Germany Roads)

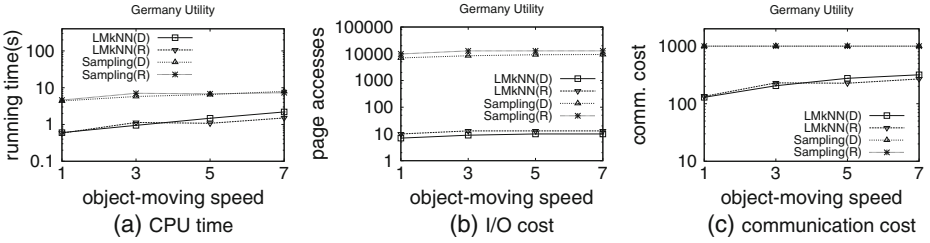


Figure 18 Analyzing the effect of the point-moving speed (Germany Utility)

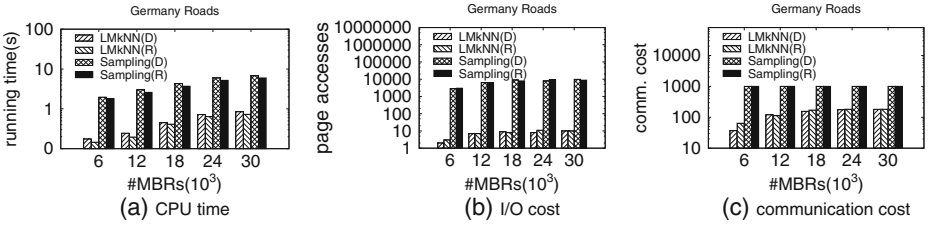


Figure 19 Analyzing the effect of the number of line segments (Germany Roads)

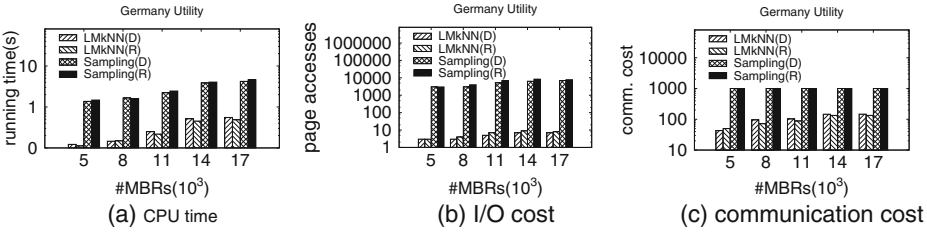


Figure 20 Analyzing the effect of the number of line segments (Germany Utility)

for LV*-Diagram, at each location, judging whether an object is in the safe region will be first executed. If the border is crossed, new safe regions will be constructed. Therefore, more sampled locations indicate more execution costs. As the number of objects locations follows a linear increase, the CPU time and communication cost increase correspondingly. Particularly, in terms of the CPU time and communication cost, our LM*k*NN outperforms *Sampling* by at least an order of magnitude. In addition, two orders of magnitude gains can be achieved with regard to I/O costs.

The effect of the object-moving speed In this suite of experiments, for every trajectory, the step length factor is varied from 1 to 7, which is used to simulate the moving speed of the query point. Figures 17 and 18 conclude our experimental results. We can find the performance of LM*k*NN is deteriorated with the increase of the moving speed. It is because the probability of moving outside the safe region is directly proportional to the speed, which determines the frequency of calling BF-*k*NN and constructing ISR. In addition, a larger velocity will destroy data access locality and thus disables the effect of buffers.

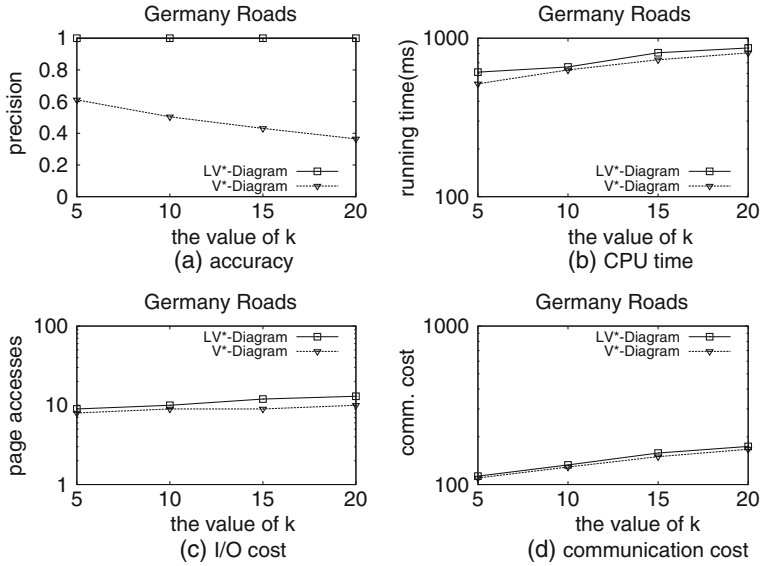


Figure 21 Comparison of V*-Diagram and LV*-Diagram

The effect of the number of data objects Finally, we evaluate the effect the data object number on the performance by sampling partial ling segments bounded by MBR uniformly from the real data sets. In Figures 19 and 20, in a fixed region, when the number of *MBR*s varies from 6,000 to 30,000 and 5,000 to 17,000 respectively, the density of line segments will be increased. And therefore, the scale of the safe region will be reduced and a larger frequency to construct a new ISR will be incurred. Also, more MBRs need to be accessed during the searching. Consequently, the CPU time, I/O costs and communication costs increase for both LM*k*NN and *Sampling*.Especially for communication costs, compared with *Sampling*, LM*k*NN is more sensitive to the density of *MBR*s due to the re-computation of the safe region. In whatever metrics, obvious performance enhancement can be achieved by leveraging the proposed ISR.

The comparison of V*-diagram and LV*-diagram Next, we test V*-Diagram (which uses $dist(q, c)$ to approximate $dist(q, s)$) and LV*-Diagram in term of efficiency and effectiveness, with k varied from 5-20. For the precision illustrated in Figure 21a, we can find the accuracy of V*-Diagram is not satisfied in our settings with the maximum error rate 60 %. From the efficiency evaluation results illustrated in Figure 21b–d, we can observe LV*-Diagram is a little inferior to V*-Diagram with regard to CPU, I/O and communication costs. One reason is that the distance evaluation pattern of LV*-Diagram is more complex than that of V*-Diagram. Another influence factor is that ISR in V*-Diagram is larger than that in LV*-Diagram because the distance will be amplified, which lead to more frequent *k*NN I/O accesses and communications from the server side. But considering the locality of consecutive *k*NN updates in the page, it is not needed to access data on disk for each *k*NN update. In general, LV*-Diagram displays a comparable performance with V*-Diagram while avoiding very serious error rates.

7 Conclusions

In this paper, we study the M*k*NN query problem over line segment data objects and present the LV*-Diagram to solve the problem. The relative geometric properties of safe regions for line segments are deeply analyzed and Line-segment-oriented M*k*NN (LM*k*NN) processing algorithms are proposed. In addition, we extend our safe regions and processing techniques to the cases of polyline segments. Experimental results show that our approach significantly outperforms the sampling based method with regard to CPU time, I/O cost and communication cost. This work is the first one to deal with the M*k*NN query by modeling the data objects as line segments so far.

Acknowledgments 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.61472071 and 61003058,and the Fundamental Research Funds for the Central Universities of China under Grant No. N130404010.

References

1. Ali, M.E., Tanin, E., Zhang, R., Kotagiri, R.: Probabilistic voronoi diagrams for probabilistic moving nearest neighbor queries. Data Knowl. Eng. **75**, 1–33 (2012)

2. Aly, A.M., Aref, W.G., Ouzzani, M.: Spatial queries with two knn predicates. Proc. VLDB Endowment **5**(11), 1100C1111 (2012)
3. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. SIGMOD Record **19**(2), 322–331 (1990)
4. Bespamyatnikha, S., Snoeyinka, J.: Queries with segments in Voronoi diagrams. Comput. Geom. **16**(1), 23–33 (2000)
5. Gao, Y., Zheng, B.: Continuous obstructed nearest neighbor queries in spatial databases. In: Proceedings of SIGMOD, pp. 577–590 (2009)
6. Gao, Y., Zheng, B., Lee, W.C., Chen, G.: Continuous visible nearest neighbor queries. In: Proceedings of 14th International Conference on Extending Database Technology, pp. 144–155 (2009)
7. Gao, Y., Zheng, B., Chen, G., Chen, C., Li, Q.: Continuous nearest-neighbor search in the presence of obstacles. ACM Trans. Database Syst. **36**(2) (2011)
8. Gao, Y., Zheng, B., Chen, G., Li, Q., Guo, X.: Continuous visible nearest neighbor query processing in spatial databases. VLDB J. **20**(3), 371–396 (2011)
9. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of SIGMOD, pp. 47–57 (1984)
10. Held, M.: VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput. Geom. **18**(2), 95–123 (2001)
11. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: Symposium on Large Spatial Databases, pp. 83–95 (1995)
12. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: Processings of SIGMOD, pp. 479C490 (2005)
13. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: Proceedings of VLDB, pp. 840–851 (2004)
14. Kulik, L., Tanin, E.: Incremental rank updates for moving query points. In: Proceedings of GIScience, pp. 251–268 (2006)
15. Kwon, H., Whang, K., Song, I., Wang, H.: RASIM: a rank-aware separate index method for answering top-k spatial keyword queries. World Wide Web **16**(2), 111–139 (2013)
16. Li, Y., Yang, J., Han, J.: Continuous k-nearest neighbor search for moving objects. In: Proceedings of SSDBM, pp. 631–642 (2004)
17. Li, C., Gu, Y., Li, F., Chen, M.: Moving K-nearest neighbor query over ob-structed regions. In: Proceedings of the 12th International Asia-Pacific Web Con-ference, pp. 29–35 (2010)
18. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: The V*diagram: A query dependent approach to moving knn queries. In: Proceedings of VLDB, pp. 1095–1106 (2008)
19. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: Analysis and evaluation of V*-kNN: an efficient algorithm for moving kNN queries. VLDB J. **19**(3), 307–332 (2010)
20. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley (1992)
21. Qi, J., Zhang, R., Wang, Y., Xue, A., Yu, G., Kulik, L.: The min-dist location selection and facility replacement queries. World Wide Web **17**(6), 1261–1293 (2014)
22. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of SIGMOD, pp. 71–79 (1995)
23. Song, Z., Roussopoulos, N.: K-nearest neighbor search for moving query point. In: Processings of SSTD, pp. 79C96 (2001)
24. Tao, Y., Papadias, D.: Time-parameterized queries in spatio-temporal databases. In: Proceedings of SIGMOD, pp. 334–345 (2002)
25. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: Proceedings of VLDB, pp. 287–298 (2002)
26. Tao, Y., Zhang, J., et al.: An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. IEEE Trans. Knowl. Data Eng. **16**(10), 1169–118 (2004)
27. Zhang, J., Zhu, M., Papadias, D., Tao, Y., Lee, D.L.: Location-based spatial queries. In: Proceedings of SIGMOD, pp. 443–454 (2003)