Continuous Monitoring of Distance-Based Range Queries

Muhammad Aamir Cheema, *Student Member*, *IEEE*, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang

Abstract—Given a positive value r, a distance-based range query returns the objects that lie within the distance r of the query location. In this paper, we focus on the distance-based range queries that continuously change their locations in a euclidean space. We present an efficient and effective monitoring technique based on the concept of a *safe zone*. The safe zone of a query is the area with a property that while the query remains inside it, the results of the query remain unchanged. Hence, the query does not need to be reevaluated unless it leaves the safe zone. Our contributions are as follows: 1) We propose a technique based on powerful pruning rules and a unique access order which efficiently computes the safe zone and minimizes the I/O cost. 2) We theoretically determine and experimentally verify the expected distance a query moves before leaving the safe zone and, for majority of queries, the expected number of guard objects. 3) Our experiments demonstrate that the proposed approach is close to optimal and is an order of magnitude faster than a naïve algorithm. 4) We also extend our technique to monitor the queries in a road network. Our algorithm is up to two order of magnitude faster than a naïve algorithm.

Index Terms—Query processing, range queries, spatial data, continuous queries, road network.

1 Introduction

We consider a set O of objects, a query point q, and a positive value r. We use $dist(o,q)$ to denote the distance between an object $o \in O$ and the query q. A distance-based range query returns every object $o \in O$ that lies within distance r of the query location q, i.e., every object such that $dist(o,q) \leq r$. Our main focus in this paper is on euclidean distance-based range queries. Since the search space around the query is a circle in this case, such queries are also called circular range queries. We also consider the case when $dist(o,q)$ is the network distance between o and q (e.g., queries moving in a road network).

Another variation of the range query, which we term "rectangular range query" (also called *window query*), returns the objects that lie within a rectangle around the query location. Distance-based range queries and rectangular range queries are inherently different and have different applications. When clear by context, we use the term *range query* to refer to the distance-based range queries.

Due to availability of inexpensive position locators, cheap network bandwidth and mobile devices with computation and storage capabilities, location-based services are gaining increasing popularity. Consequently, continuous monitoring of spatial queries has received significant research attention in the past few years [1], [2], [3], [4], [5], [6], [7], [8].

In this paper, we study the continuous monitoring of moving range queries over static data objects, i.e., a scenario where the queries are constantly moving whereas the data objects do not change their locations. Such scenario has many interesting applications. Consider the example of a family travelling by car. Suppose they need to reach their final destination by a certain time and only have up to 90 minutes available for lunch. They may want to continuously monitor restaurants within 10 km of their current location so that they can choose a restaurant that serves their favorite meals, and will not take more than 15 minutes to reach. As another example, a bomber plane might want to continuously monitor the enemy targets (e.g., airport, arms depot) that are within its attack range.

We next discuss two models to monitor spatial queries.

Client-server model. In this model, the clients issue queries and the central server is responsible for the computation of these queries. For example, a person walking down the street may issue a query to his mobile service provider to continuously report the coffee shops within 1 km of the issuer's location. It may be assumed that the server processes the query in the main-memory, i.e., the data objects are stored in the main-memory along with other relevant information needed to efficiently update the results. However, such systems require that the server continuously maintains this information in the main-memory in order to provide the service.

We neither require that the data objects are stored in the main-memory nor do we maintain any query information in the main-memory. One advantage of this is that the service can be run on demand. Since the objects are stored in the secondary memory and no main-memory information is maintained, the server can go to sleep mode if there is no

- *M.A Cheema, X. Lin, W. Zhang, and W. Wang are with the School of Computer Science and Engineering, University of New South Wales, NSW 2052, Australia.*
E-mail: {macheema, lxue, zhangw, weiw}@cse.unsw.edu.au.
- *L. Brankovic is with the School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan NSW 2308, Australia.*
E-mail: ljiljana.brankovic@newcastle.edu.au.

Manuscript received 20 Mar. 2010; revised 10 Aug. 2010; accepted 3 Sept. 2010; published online 9 Dec. 2010.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2010-03-0168.
Digital Object Identifier no. 10.1109/TKDE.2010.246.

query. When a query arrives, the server computes the results and the *safe zone*, which are then sent to the client. The safe zone is an area such that the reported results are valid as long as the client (i.e., query) remains within the safe zone. A query that leaves its safe zone sends an update request. The server updates the safe zone and the results, and sends them back to the client.

Local computation model. In the first application mentioned above, the car may have a GPS navigation system with points of interest (e.g., restaurants) stored in its memory card. Since the navigation systems have limited main-memory and computational capacity, it may be challenging to compute the results of the range query whenever the query changes its location (the car is continuously moving). Our proposed approach returns a safe zone which guarantees that the results of the query do not change as long as the query remains within the safe zone. The safe zone is updated efficiently when the query leaves the safe zone. Our experimental results demonstrate that the overhead to compute the safe zone is small compared to the cost of the range query. This enables our framework to work effectively on the devices with limited main-memory and computation power. We next highlight some advantages of our proposed approach.

1. The computation of the safe zone reduces the overall computation time because the query needs to be reevaluated only when it leaves the safe zone. Our experiments indicate that the cost of computing the safe zone is small compared to the cost of the range query.
2. Although the shape of the safe zone may be arbitrarily complex, we can still efficiently check whether the query lies within it. If the query is based on network distance, the safe zone itself is a small network that is a subset of the original network and it has to be determined whether the query lies within the safe zone or not. For the circular range queries, we utilize the fact that the safe zone only depends on the so-called *guard* objects. Checking whether the query lies within the safe zone takes k distance computations, where k is the number of guard objects. Our experimental results demonstrate that the average number of guard objects is around five. This makes our proposed approach applicable for the clients that have limited computational power. We also present a theoretical analysis and give an upper bound on the expected number of guard objects for the queries with the diameter of the safe zone no more than a constant times its expected value.
3. We do not require the data objects to be stored in the main-memory, which allows our approach to work on systems with limited main-memory (e.g., GPS navigation systems).
4. When an update request is received, the server computes the new safe zone and the results for the circular range queries. After updating the results, the server only sends new information to the clients. For example, if the client was informed that an object o_i is within its range, the object o_i is not sent again in updated results if it still lies within the range. If in the future such object o_i ceases to be within the range, the client is informed that o_i is out of the range. Our experimental results demonstrate that this significantly reduces the amount of data transmitted from the server to the clients.
5. In the client-server paradigm, our proposed approach does not require the server to maintain or record any information related to the queries, yet it efficiently updates the safe zones. This enables the server to run this service on demand.

Note that some computation models require queries to get registered at the server and report their locations after every t time units. Our approach can be readily applied to such systems. In the rest of the paper, we assume a model where a query contacts the server only if it leaves the safe zone.

Although there exists a safe zone-based solution for moving window queries [6], this technique is not applicable to the moving circular range queries. In Section 2, we show that it is not possible to extend this technique to the case of the distance-based range queries as the problems of monitoring moving window queries and the distance-based range queries are inherently different. We apply an aggressive approach to prune the objects/entries that cannot affect the results and/or the safe zone. Our pruning rules are tight and the performance of our solution is close to optimal.

We next summarize our contributions in this paper.

- We present an efficient and effective technique to monitor the moving circular range queries by adopting the concept of safe zones.
- We present a rigorous theoretical analysis to verify the effectiveness of our safe zone-based approach for the moving circular range queries. More specifically, we evaluate the probability that a query moves out of the safe zone within one time unit, the expected distance it travels before it leaves the safe zone, and an upper bound (which is a constant) on the expected number of guard objects for the queries with the diameter of the safe zone no more than a constant times its expected value. Our experimental results confirm the accuracy of the presented theoretical analysis.
- We conduct extensive experiments to show the effectiveness of our approach. We compare our algorithm with an optimal solution and a naïve solution. The experimental results indicate that our proposed approach is close to the optimal solution and an order of magnitude faster than the naïve algorithm.
- Based on nontrivial access order and pruning rules, we present a complete framework for answering the distance-based range queries in a road network. Experiments demonstrate that our algorithm is up to two orders of magnitude faster than a naïve algorithm.

The remainder of the paper is organized as follows: in Section 2, we give an overview of the related work. We introduce our framework and pruning rules for processing the moving circular queries in Section 3. In Section 4, we present our safe zone-based solution to the moving circular range queries. Theoretical analysis is presented in Section 5.

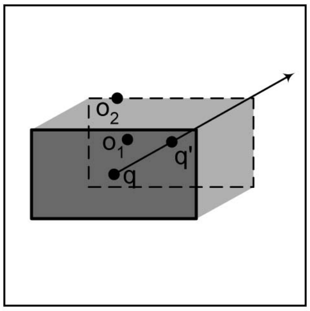


Fig. 1. A time-parameterized window query.

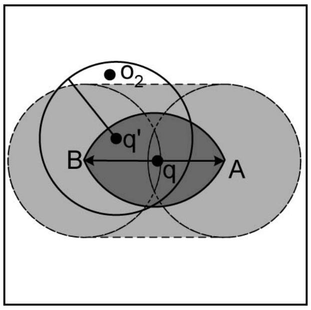


Fig. 2. TP circular queries cannot be used to construct safe zone.

In Section 6, we present our techniques to answer moving range queries in a road network. The experimental results are reported in Section 7. Section 8 concludes the paper.

2 Related Work

2.1 Spatial Queries in Euclidean Space

Continuous monitoring of spatial queries has been extensively studied in the recent past [9], [10], [2], [11], [3], [12], [13], [6]. Prabhakar et al. [14] proposed velocity constrained indexing and query indexing for continuous evaluation of static queries over moving objects. Mokbel et al. [15] introduced an algorithm (SINA) for evaluating a set of concurrent spatial queries, which reduces the overall cost by shared execution and incremental evaluation.

Several distributed processing techniques to continuously monitor range queries have also been proposed [7], [1], [16], [17]. Gedik and Liu [7] introduce a technique called MobiEyes, which reduces the computation load on the server and communication costs between the clients and the server by delegating some computation load to the client objects (e.g., mobile devices). In [18], the authors propose a motion adaptive indexing scheme that uses the concept of motion sensitive bounding boxes to model moving objects and queries. Hu et al. [2] propose a generic framework to monitor continuous range queries and kNN queries over moving objects. They define the safe zones for each object such that the query results remain unchanged if the object does not leave the region. However, their approach is not designed for moving queries. Wu et al. [19] use a new query indexing method called CES-based indexing to minimize the total query evaluation time.

We now present the related techniques that are specifically designed for moving spatial queries. Several techniques have been proposed to construct safe zones for moving kNN queries [20], [21], [6], [22], [23] and moving window queries [6]. However, to the best of our knowledge, there does not exist any safe zone-based technique to continuously monitor moving circular range queries. We next show that the existing work cannot be extended to monitor moving circular range queries continuously.

Tao and Papadias [8] introduce Time-Parameterized queries (TP queries). A TP query assumes that the motion pattern (e.g., path and speed) of the query is known and retrieves the current results along with a future time at which the current results will become invalid. A TP query also reports the object that invalidates the results. In [8], the techniques to answer TP kNN queries, TP window queries and TP join queries are presented.

Fig. 1 shows an example of a window query where the current location of the query is q and its window is shown with a solid line (the search space is shown in a dark shade). The current result of the window query q is the object o_1. A TP window query is issued to find the object that invalidates the current result when the query is moving in the direction shown by the arrow. The query returns the object o_2 as it invalidates the current result when the query reaches the location q'. In other words, when the query reaches q', it has objects $o1$ and o_2 within its window and not only $o1$. The minimal area searched by the TP query is shown shaded in Fig. 1.

Based on TP queries, Zhang et al. [6] present a solution to continuously monitor kNN queries and the window queries. They use TP queries to identify the safe zones for moving queries. The algorithm starts by assuming that the whole space is the safe zone. TP queries are then issued toward the corners of the current safe zone. If a TP query retrieves an object that has not already been considered, the safe zone is trimmed using that object (for details, see [6]); otherwise, the corner is marked as confirmed. The algorithm terminates when all the corners are confirmed.

We note that there does not exist any reported work on TP circular range queries and the technique presented in [6] cannot be applied to such queries. Even if the technique to answer TP window queries are extended to answer the TP circular range queries, the TP circular range queries cannot be used to construct the safe zone. The reason is as follows: the key observation used in the technique presented in [6] is that if none of the TP queries issued toward corners of a region returns a new object, the region is guaranteed to be the safe zone. This observation does not hold for the moving circular range queries. Consider the example in Fig. 2 where the current region is shown dark shaded. The TP range queries are issued toward each of the two corners A and B and they search the space shown shaded in the figure. No object is returned by either of the TP range queries. However, the region cannot be guaranteed to be the safe zone. Consider that the query moves to the location q'. Then the object o_2 lies within its range, which invalidates the results.

2.2 Spatial Queries in Road Networks

Significant research attention has been given to developing techniques for spatial queries in road networks. kNN queries [24], [25], [26], [27], [28], [29], [30], [31] and range

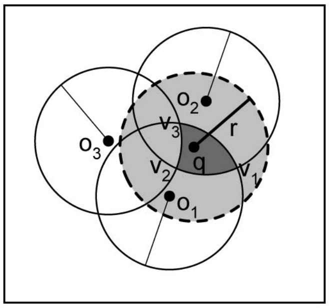


Fig. 3. A range query and its safe zone.

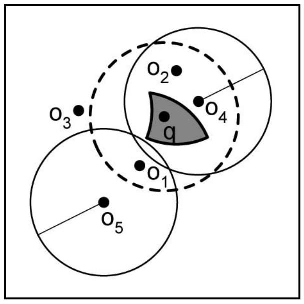


Fig. 4. Some objects do not affect the safe zone.

queries [32], [28], [33], [34] are among the most studied spatial queries in road networks. Chen et al. [35] study the path k-NN queries that returns kNNs with respect to the shortest path connecting the destination and the user's current location.

Papadias et al. [28] propose a framework to support nearest neighbor queries, closest pairs queries, range queries, and distance joins on a road network. However, they assume that the queries and the objects have fixed positions in the spatial network. Wang and Zimmermann [34] propose a solution to answer static range queries over moving objects. They utilize a disk resident R-tree to store the network and a grid structure to store the positions of moving objects. The main idea is to first find the edges that may contain the objects within the range and then the grid cells that overlap with the edges are used to retrieve the objects. Liu et al. [33] present a distributed processing technique to solve the moving range queries over moving objects. Their approach relies on the computation power of the moving objects and each moving object reports to the server when it affects the results of one or more queries. Stojanovic et al. [32] propose technique for continuous monitoring of range queries over moving objects. The range of the query may be defined by a user selected area, a map window, a polygon, a circle, or a part of the road segment.

Kriegel et al. [36] study the problem of proximity monitoring in road networks. Given a proximity threshold ϵ and a set of moving objects, a server responsible for proximity monitoring continuously reports the pairs of objects that are within a distance ϵ to each other. Küpper and Treu [37] propose a technique for the same problem in euclidean space. Both of the techniques assign each moving object a region such that as long as the object remains within this region it does not need to report its location to the server. Note that these techniques can be adopted to answer the distance-based range queries by setting the proximity threshold to r and considering only the pairs of objects that contain the query object q. However, the focus of these techniques is to reduce the communication cost between the moving objects and the server. On the other hand, the focus of our technique is to minimize the computation time. Moreover, our framework is suitable for both the client-server model and the local computation model.

3 Framework

3.1 Solution Overview

Consider the example in Fig. 3 where a range query q is shown. Its range is r and the area within its range is shown shaded. Some objects around it are also shown. The objects that lie within the range form the result set and are called *internal* objects (e.g., the objects o_1 and o_2). The objects that do not lie within the range are called *external* objects (e.g., the object o_3). Let C_i be a circle of radius r with center at the location of the object o_i. Fig. 3 shows the circles for the objects o_1, o_2 and o_3.

Note that all the internal objects contain q in their circles whereas the external objects do not. An internal object o_i ceases to be within the range only when the query q leaves its circle C_i. Similarly, an external object becomes included in the result only if the query enters its circle. In other words, the result of the query q does not change as long as q does not leave or enter any circle. Hence, the safe zone of a query q is defined by the boundaries of the circles around it. In the example in Fig. 3, the dark shaded area is the safe zone because q does not enter or leave any circle as long as it remains in this area. Formally, safe zone S can be defined as the intersection of the circles of internal objects minus the circles of external objects. That is, $S = \cap_i C_i - \cup_j C_j$ for every internal object o_i and every external object o_j.

Please note that as we consider new objects in order to calculate the safe zone, we may find that some objects may not affect the shape of the safe zone. Consider the example in Fig. 4 where the objects o_4 and o_5 are shown. The circle of the internal object o_4 completely contains the current safe zone[1] of q. Hence, it does not change the shape of the current safe zone and will not define the final safe zone. Similarly, the circle of the external object o_5 does not intersect the current safe zone and consequently does not affect its shape. For this reason, the final safe zone can be defined without using the circles of o_4 and o_5. In this paper, the objects that contribute to the shape of the final safe zone are called *guard* objects (e.g., o_1, o_2 and o_3). An internal (external) object that contributes to the final safe zone is called an internal (external) guard. Internal guards in this example are o_1 and o_2 whereas o_3 is an external guard. For the sake of simplicity, in what follows we refer to both "current safe zone" and "final safe zone" simple as "safe zone."

1. We use the term current safe zone because the safe zone is being constructed and is not the final safe zone. From now on, the current safe zone is called safe zone and the current guard objects are called guard objects when there is no ambiguity.

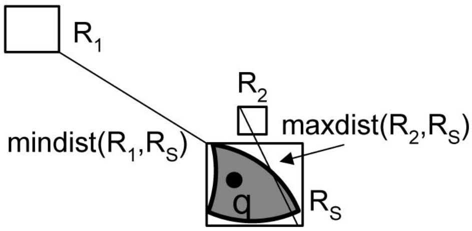


Fig. 5. Pruning using the approximation of safe zone.

3.1.1 Data Structure at a Glance

All objects are indexed by a disk-resident R-Tree [38]. For each query, the server keeps the following information in its memory during the computation of the safe zone: 1) its location; 2) the list of internal objects called *answer list*; 3) the list of guard objects. For each guard object, the server stores its arcs that contribute to the safe zone. In the example in Fig. 3, the object o_1 has an arc with two end vertices v_1 and v_3. We use this arc (or vertices) for effective pruning. Note that the server stores this information in its memory only during the construction of the safe zone, and discards this information after the safe zone has been computed and sent to the client.

3.1.2 Checking whether q Lies in the Safe Zone

Since the clients that issue queries (e.g., mobile devices) have limited computational power, it is desirable that checking whether the client is inside the safe zone is not computationally expensive. Although the shape of a safe zone may be complex, the cost of checking whether q lies in the safe zone takes only k distance computations where k is the number of guard objects. More specifically, the query q computes its distance from each of the guard object. If it lies within the circle of every internal guard and lies outside the circle of every external guard then it lies within the safe zone. Our experimental results show that the average number of guard objects is around five. We also present a theoretical analysis to give an upper bound on the expected number of guard objects for the queries that satisfy certain constraints.

A simple approach to compute the safe zone is to consider all objects and find the objects that actually contribute to the safe zone. However, the number of objects that are considered must be reduced in order to reduce the I/O cost and to improve the CPU time. We next present five effective pruning rules that significantly reduce the number of considered objects.

3.2 Pruning Rules

As shown in the example in Fig. 4, some objects do not affect the safe zone. More specifically, if the circle of an object contains the safe zone (such as o_4 in Fig. 4) or lies completely outside the safe zone (such as o_5 in Fig. 4), that object does not affect the shape of the safe zone. In this section, we present some effective pruning rules to prune such objects. Note that only the circles of internal objects may contain the safe zone and only the circles of external objects may completely lie outside the safe zone. Hence, some pruning rules are specific to the internal objects and some are to be applied only on external objects.

First, we present pruning rules based on the approximation of the safe zone by a rectangle. Let a and b be two rectangles or points; we use $mindist(a, b)$ and $maxdist(a, b)$ to denote the minimum and maximum distances between them, respectively.

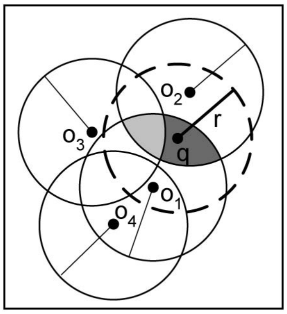


Fig. 6. Illustration of pruning rule 3.

3.2.1 Using Approximation of the Safe Zone

Let R_S be the minimum bounding rectangle of the current safe zone as shown in Fig. 5. Let R_{cnd} be a rectangle that contains some candidate objects.

PRUNING RULE 1: If $maxdist(R_{cnd}, R_S) < r$ then no object in R_{cnd} can affect the safe zone.

Proof. Let o be an object in R_{cnd}. For every point $p \in R_S$, $dist(o, p) < r$ because $maxdist(R_{cnd}, R_S) < r$. Hence, the circle of o contains *every* point p of the safe zone, i.e., o does not affect the safe zone. □

PRUNING RULE 2: If $mindist(R_{cnd}, R_S) > r$ then no object in R_{cnd} can affect the safe zone.

Proof. Let o be an object in R_{cnd}. For every point $p \in R_S$, $dist(o, p) > r$ because $mindist(R_{cnd}, R_S) > r$. Hence, the circle of o does not contain *any* point p of the safe zone, i.e., o does not affect the safe zone. □

In the example of Fig. 5, where $maxdist(R_2, R_S) < r$, it can be immediately verified that any object in R_2 contains the safe zone in its circle. Similarly, $mindist(R_1, R_S) > r$ and every object in R_1 can also be pruned.

3.2.2 Using the Guard Objects

Although the rectangle-based pruning is inexpensive, it is unfortunately not very tight. We present tighter pruning rules below, based on the positions of the guard objects.

PRUNING RULE 3: If $mindist(R_{cnd}, o_i) > 2r$ for *any* internal guard object o_i then no object in R_{cnd} can affect the safe zone.

Proof. An object can only affect the safe zone if its circle intersects the safe zone. Safe zone is the area defined by the intersection of the circles of the internal guard objects minus the circles of the external guard objects. Hence, the circle of any internal guard object contains the whole safe zone. Thus a circle can only intersect the safe zone if it intersects the circles of *all* internal guard objects. Consequently, if an object o_j lies at a distance greater than $2r$ from any internal guard o_i, it cannot intersect the safe zone. □

In Fig. 6, the object o_4 cannot affect the safe zone because it lies at a distance greater than $2r$ from o_2. To show the area

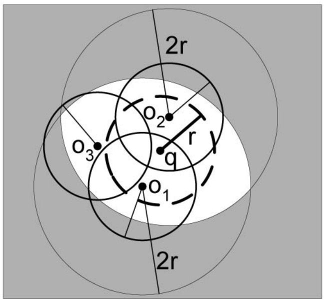


Fig. 7. Area pruned by the rule 3.

that is pruned by this pruning rule, we zoom out Fig. 6 and show the pruned area in Fig. 7. The shaded area can be pruned because every point in it lies at a distance greater than $2r$ from at least one of o_1 and o_2. This pruning rule prunes the rectangles that contain external objects.

Before we present tighter pruning rules, we provide few auxiliary observations and lemmas.

Consider a circle C with center at M and radius r, and any point E in the plane (inside or outside the circle) (see Fig. 8). The line that passes through E and M intersects the circle at two points, A and B. Without loss of generality, we assume that $dist(A, E) < dist(B, E)$, as shown in Fig. 8. We make the following observation.

Observation 1. *Let C be a circle of radius r, and M, E, A, and B be the points as described above. The distance between E and any point D on the circle monotonically increases as D moves along the circle from point A to B, either clockwise or counterclockwise. In other words, any point D' that lies before D while travelling on the circle from A to B satisfies $dist(E, D') < dist(E, D)$.*

The above observation can be easily verified from the triangle $\triangle EMD$. If we denote $\overline{MD}$ by r and the length of $\overline{EM}$ by x, then the length of $\overline{DE}$ is given by the law of cosine as $dist(D, E) = \sqrt{r^2 + x^2 - 2rx \cdot cos(\angle EMD)}$. Note that as D travels along the circle from A to B, the angle $\angle EMD$ increases from 0 to 180 degree and its cosine monotonically decreases from 1 to -1. As both r and x remain unchanged, the distance $dist(D, E)$ monotonically increases. Note that we do not require x to be smaller than r, so the observation also holds for the case when E lies outside the circle.

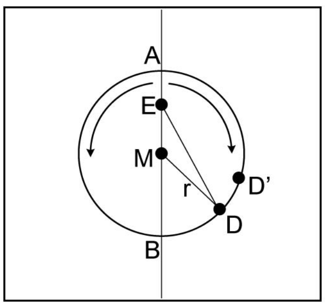


Fig. 8. Observation 1.

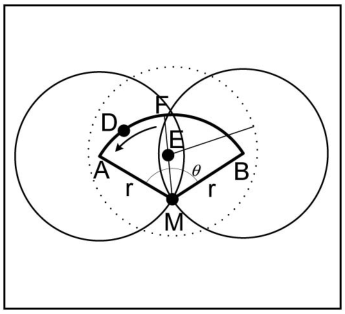


Fig. 9. Lemma 1.

Based on Observation 1, we present the following lemma that is used in our next pruning rule.

Lemma 1. *Let $\widehat{AB}$ be an arc of radius r with subtending angle $\theta < 180$ degree where A and B are the end points of the arc and M is the center (as shown in Fig. 9). Let C_A and C_B be two circles of radius r centered at A and B, respectively. Every point E that lies inside both the circle C_A and circle C_B satisfies the following: the circle of radius r with center at E (the dotted circle in Fig. 9) contains every point of the arc $\widehat{AB}$.*

Proof. In order to prove the lemma, we need to show that the distance of E from any point D that lies on the arc $\widehat{AB}$ is smaller than r. If we extend the line joining M and E, it cuts the arc at point F which is the minimum distance from E to the circle. We prove the lemma for the arc $\widehat{AF}$ and the proof for the arc $\widehat{FB}$ is similar. By Observation 1, we know that any point D that lies on the arc $\widehat{AF}$ satisfies $dist(E, D) \leq dist(E, A)$. As the point E lies inside the circle C_A, $dist(E, A) < r$. Hence, $dist(E, D) < r$ for any point D. □

Please note that the lemma does not hold if the subtending angle $\theta \geq 180$ degree as the line joining M and E intersects the arc $\widehat{AB}$ at point F which is the maximum distance from E to the circle and is greater than r (Fig. 10).

Based on Lemma 1, we present a pruning rule to prune the rectangles that contain internal objects.

PRUNING RULE 4: Let S be a safe zone such that every arc that defines it has subtending angle smaller than 180 degree. If $maxdist(R_{cnd}, v_i) \leq r$ for every vertex v_i of the safe zone S, then no object in R_{cnd} can affect the shape of the safe zone.

Proof. Let E be a point that lies within all the circles of radius r centered at vertices of the safe zone. From

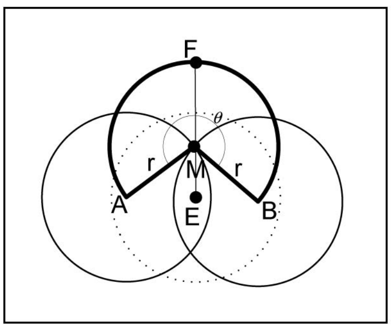


Fig. 10. When $\theta > 180$ *degree*.

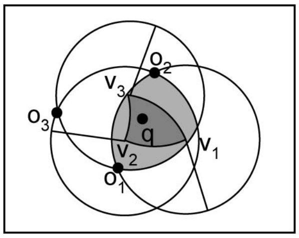


Fig. 11. Pruning rule 4.

Lemma 1, we know that the circle centered at E contains every arc of the safe zone. Hence, it contains the whole safe zone and cannot affect its shape. □

Fig. 11 shows three circles of radius r with centers at the vertices v_1, v_2, and v_3. Any object or rectangle that lies in the shaded area can be pruned because its distance to any vertex cannot be greater than r.

For our final pruning rule, we need the following lemma.

Lemma 2. *Let $\widehat{AB}$ be an arc with center at M, radius r and subtending angle $0 < \theta < 360$ degree as shown in Fig. 12. The distance of E from every point of the arc $\widehat{AB}$ is greater than r, if E satisfies either of the following conditions:*

1. E lies within the angle range θ and $dist(E, M) > 2r$;
2. E lies outside the angle range θ, $dist(E, A) > r$ and $dist(E, B) > r$.

Less formally, if E lies within the shaded area in Fig. 12, its distance to any point on the arc $\widehat{AB}$ is greater than r.

Proof. We first consider a point E_1 that lies within the angle range θ (see Fig. 12). We draw a line through points E_1 and M and we denote the intersection of the line and the arc by G. By Observation 1 $dist(E_1, G)$ is the minimum distance from the point E to the arc $\widehat{AB}$. Since $dist(E_1, M) > 2r$, it follows that $dist(E_1, G) > r$ and thus $dist(E_1, D) > r$ for any point D on the arc $\widehat{AB}$.

We now consider a point E_2 that lies outside the angle range θ (see Fig. 12). Again, by Observation 1, the minimum distance from E_2 to the circle is $dist(E_2, F)$ (see Fig. 12), and the distance between E_2 and the points on the circle increases monotonically as we move along the circle away from the point F. Thus for every point D on the arc $\widehat{AB}$ we have either $dist(E_2, D) \geq dist(E_2, A) > r$ or $dist(E_2, D) \geq dist(E_2, B) > r$. □

Based on Lemma 2, we present our final pruning rule that prunes external objects.

PRUNING RULE 5: No object in a rectangle R_{cnd} can affect the safe zone if R_{cnd} satisfies Lemma 2 (i.e., R_{cnd} lies completely in the shaded area of Fig. 12) for every arc of the safe zone.

Proof. The proof immediately follows from Lemma 2 as any point in R_{cnd} has minimum distance to the boundary of the safe zone greater than r. Hence, its circle cannot intersect the safe zone. □

In order to apply this pruning rule, we check the minimum distance of the rectangle R_{cnd} from M, A, and B. If the rectangle completely lies outside the angle range θ, it can be pruned if its minimum distance from both A and B is greater than r. Otherwise, it can be pruned if its minimum distance from M is greater than $2r$.

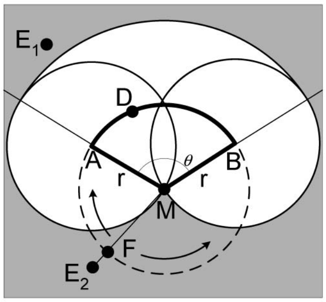


Fig. 12. Illustration of Lemma 2.

Fig. 13 shows the area pruned by the rules 4 and 5, where the outer shaded area is pruned by the pruning rule 5 and we call it *external* pruned area. The inner shaded area is pruned by the rule 4 and we call it *internal* pruned area.

The arguments similar to those used in proofs of Lemma 1 and 2 can be used to show that the pruning rules are tight. In other words, any object that lies in the unpruned area (the white area in Fig. 13) affects the shape of the current safe zone. Note that although the rectangle-based pruning rules have less pruning power, they are important because they are computationally less expensive. We first apply the rectangle-based pruning rules and if an object is not pruned, we apply the guard objects based pruning rules.

4 Technique

Initially, the whole space is assumed to be the safe zone. We then access each object that cannot be pruned, and use its circle to trim the safe zone. The algorithm stops when all the objects that cannot be pruned are accessed. The order in which the objects are accessed is important as better access order retrieves fewer objects that affect the safe zone. We first present our proposed access order. Second, we present our query-processing algorithm followed by the algorithm to trim the safe zone. Finally, we present an efficient technique to update the safe zone when the query leaves it.

4.1 Access Order

After applying the pruning rules presented above, there may be several objects left in the unpruned area. The order in which these objects are accessed is important. Intuitively,

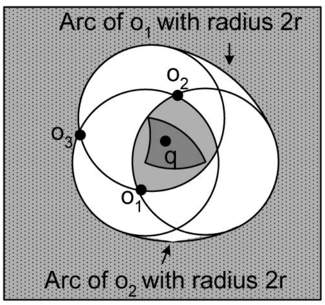


Fig. 13. Pruning by rules 4 and 5.

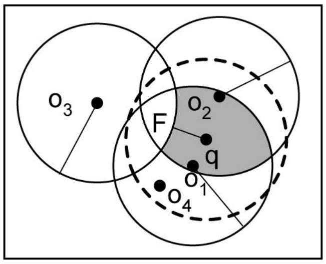


Fig. 14. Importance of access order.

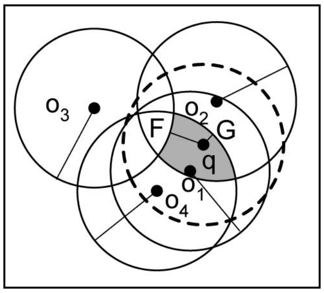


Fig. 15. o_1 is not a guard object anymore.

the objects that lie closer to the boundary of the range query have a more significant effect on the shape of the safe zone and should be accessed first.

Consider the example in Fig. 14, where the boundary of q is shown in thick broken line. The objects o_1, o_2, and o_3 are accessed first and are the current guard objects. The object o_4 that lies closer to the boundary than all of the existing guard objects is guaranteed to affect the shape of the safe zone. In Fig. 15, the object o_4 is accessed and the safe zone is shown after trimming with respect to its circle. We present a lemma that shows the importance of the objects located near the boundary for constructing the safe zone.

Lemma 3. *Let o_i be an object that is closer to the boundary of the range query than all current guard objects. The object o_i is guaranteed to affect the shape of the current safe zone.*

Proof. Without loss of generality, consider the example in Fig. 14 where the current safe zone is shown shaded. The closest guard object to the boundary of the range query is o_3. Thus the minimum distance from the query to the current safe zone is $|dist(o_3, q) - r|$. Any object o_4 that lies closer to the boundary than o_3 has a point G on its circle with distance $|dist(o_4, q) - r|$ from the query, which is less than $|dist(o_3, q) - r|$ (see Fig. 15). Hence, the circle of o_4 has at least one point inside the current safe zone so it affects the safe zone. □

In fact, in this particular example, the object o_4 is not only a guard object but it also removes the object o_1 from the list of the guard objects. Consider Fig. 15, where the object o_4 has been considered for trimming and the new safe zone is shown shaded after. Clearly, the circle of the object o_1 does not contribute to the safe zone anymore, and consequently o_1 is removed from the list of the guard objects. This example supports the intuition that the objects that lie closer to the boundary of the query should be accessed first. Our experimental results demonstrate the effectiveness of this proposed access order (Fig. 31 in Section 7). Next, we present an efficient algorithm that accesses the objects in the proposed order.

4.2 Algorithm

We use an R-Tree [38] to index the objects. Each leaf and index node of an R-tree contains pointers to its entries and a minimum bounding rectangle that contains all its objects. For details, please see [38].

Algorithm 1 outlines the solution. A min-heap is initialized with the root entry of the R-tree. The entries are deheaped iteratively until the heap becomes empty. If a deheaped entry e has $maxdist(e, q) < r$, then all the objects in it are internal and we apply pruning rules 1 and 4. If the entry is pruned, we do not need to check any objects within it for the construction of the safe zone. However, as these objects are internal, they contribute to the answer to be sent to the query. Therefore, we insert all the objects that are within this entry to the answer list (lines 4-7).

Algorithm 1. Range Query (q, r)

```
Input:    q: the query point; r: range of the query;
Description:
 1: initialize a min-heap H with root of the R-Tree
 2: while H is not empty do
 3:    deheap an entry e
 4:    if maxdist(e, q) < r then
 5:       if pruned using rules 1 and 4 then
 6:          insert all objects of e in the answer list
 7:          continue
 8:    else if mindist(e, q) > r) then
 9:       If pruned using rules 2, 3 and 5, continue;
10:    if e is an object then
11:       TrimSafeZone(e,q,S)  /* Algorithm 2 */
12:       if e is an internal object, insert in the answer list
13:    if e is a leaf or index node then
14:       for each entry c in e do
15:          insert c into H with key set to its minimum distance
             from boundary
16: send guard objects and answer list to the query q
```

If the deheaped entry e has $mindist(e, q) > r$, then all the objects in it are external objects and we apply pruning rules 2, 3, and 5 (lines 8 and 9). If the entry is pruned, we continue the algorithm by deheaping the next entry. Note that an entry e for which $mindist(e, q) \leq r \leq maxdist(e, q)$ cannot be pruned by any of the pruning rules. This is because such entries may contain both internal and external objects, while all the proposed pruning rules are applicable either to internal objects or to external objects. For this reason, we do not consider such entries for pruning.

If e is an object and cannot be pruned, we use it to trim the safe zone; if it is an internal object, we also insert it into the answer list (lines 10-12). Otherwise, if e is a leaf or index node, we insert its entries into the heap with key of each entry set to minimum distance of the entry from the boundary of the range query (lines 13-15). The algorithm stops when the heap becomes empty.

The minimum distance of an entry e from the boundary of the range query is computed as follows: if $mindist(e, q) \leq r$ and $maxdist(e, q) \geq r$, then the minimum distance of this

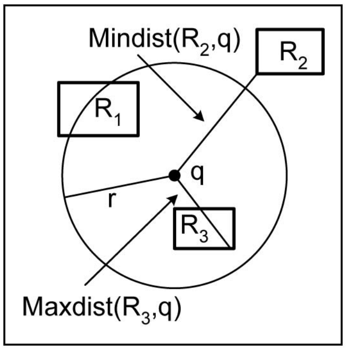


Fig. 16. Minimum distance from the boundary.

entry from the boundary is zero because the entry e overlaps the boundary (see R_1 in Fig. 16). If $mindist(e, q) > r$, then the minimum distance of this entry is $mindist(e, q) - r$ (see R_2 in Fig. 16). Finally, if the $maxdist(e, q) < r$ then the minimum distance is $r - maxdist(e, q)$ (see R_3 in Fig. 16).

In a special case when there is no object within the range, the whole space minus the circles of all the external objects will be the safe zone. However, the number of guard objects may be arbitrarily large. For such cases, in order to restrict the space, we treat query location as a virtual internal object. Then only the objects within distance $2r$ of the query may be the guard objects.

4.3 Trimming the Safe Zone

Algorithm 2 shows the procedure to trim the safe zone with respect to an object o. Note that to trim the safe zone, we only need to update the guard objects and the vertices of the safe zone and we do it as follows: for each guard object o_i, the intersection points of the circles of o and o_i are computed. If the intersection point lies on the boundary of the safe zone, the point is added as the vertex of the safe zone (lines 1 to 3). Then, the object o is added as the guard object.

Algorithm 2. TrimSafeZone (o, q, S)

Input: o: an object o to be used for updating the safe zone; q: the query point; S: the list of current guard objects;
Description:
1: **for** each guard object o_i in S **do**
2: **for** each intersection point v_i of circles of o and o_i **do**
3: add v_i to vertices list if v_i lies on the boundary of the safe zone
4: add o to the list of guard objects S
5: **if** o is an internal object **then**
6: remove every vertex v if $dist(o, v) > r$
7: **else if** o is an external object **then**
8: remove every vertex v if $dist(o, v) < r$
9: remove every guard object o from S if all its related vertices have been removed

Finally, the existing vertices that are no longer in the safe zone are removed and the objects that no longer have any associated vertices are removed from the list of guard objects (lines 5 to 9).

Fig. 17 illustrates the Algorithm 2 and shows the safe zone (shaded), together with its current guard objects o_1, o_2, and o_3. The safe zone is to be trimmed by a new object o_4. For the sake of clarity, the circles of o_1 and o_3 are not shown. The circle C_4 of the object o_4 intersects the circle C_2 of the object o_2 at two points, v_4 and v_5. The intersection point v_4 lies on the boundary of safe zone, so it is added to the list of vertices of the current safe zone. The intersection point v_5 lies outside the safe zone so it is deleted. Similarly, the intersection points of the circle C_4 with the circles of o_1 and o_3 are considered and v_6 is added to the list of vertices. All other intersection points lie outside the safe zone and are deleted.

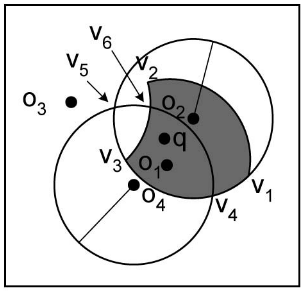


Fig. 17. Illustration of the trimming (Algorithm 2).

Now the vertices of the safe zone that are not valid anymore are to be deleted. Since o_4 is an internal object (it contains q in its circle), all vertices that lie outside its circle are deleted. For this reason, the vertices v_1 and v_2 are deleted. The related object o_1 is also deleted as it no longer has any associated vertex. After trimming of the safe zone, its vertices are v_3, v_4, and v_6 and the guard objects are o_2, o_3, and o_4.

4.4 Updating the Safe Zone when Query Leaves It

When the query leaves its safe zone, it sends its current location and current guard objects to the server. The server updates the answer list (the list of internal objects), computes the new safe zone and sends it to the query. A straightforward approach is to compute the safe zone and answer list from scratch. However, this is not only expensive but can also cause a large amount of data to be transmitted from the server to the query if the answer list contains a large number of objects.

In this section, we propose an effective approach to update the safe zone and the answer list, called *smart-update*. The smart-update utilizes the previous safe zone of the query and avoids searching the area that was visited before. Furthermore, instead of computing and sending all the objects lying within the range, the smart-update sends a list of objects called *delta list* that contains two types of objects. An object o_i^+ indicates that the object o_i that was previously external is now internal. So, the client must add it in its answer list. An object o_i^- indicates that the object o_i that was previously internal is now external. Hence, the client must remove it from its answer list.

Fig. 18 shows that a query q leaves the safe zone and moves to q'. The shaded area corresponds to the area that was pruned with respect to its previous safe zone. The smart-updates first considers the existing guard objects and constructs an initial safe zone (as shown in Fig. 19). Then, the smart-update uses two observations to reduce the search area. 1) The white area of the Fig. 18 cannot contain any object. The proof is straightforward because if there were any object in the white area, it would have affected the previous safe zone. Hence, the smart-update does not

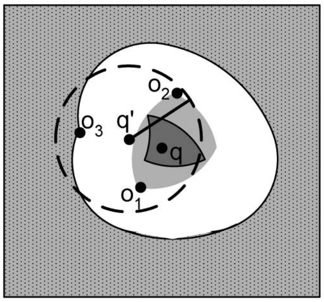


Fig. 18. q leaves the safe zone.

search this area. 2) The query q contains in its answer list all the objects that are in the internal pruned area (the internal shaded area of Fig. 18). Hence, the objects that lie within distance r from q' and lie in the internal pruned area are not required to be sent to the client.

In the example in Fig. 19, the object o_4 is not sent to the query because it lies in the previous internal pruned area and the query already contains it. However, the object o_5 must be sent so that the query removes it from its answer list.

We remark that our approach can be easily adopted to update the safe zone for the case when the underlying objects issue updates (e.g., an object appears or disappears from the data set). Due to the space limitations, we omit the details and refer the readers to Section 6 of [39].

5 Theoretical Analysis

In this section, we present a theoretical analysis to evaluate the effectiveness of the safe zone. In what follows we assume that there are N objects in total and that they are uniformly distributed in a square unit universe.

5.1 Escape Probability (P_{esc})

We first analyze the *escape probability* P_{esc}, which we define as the probability that a query q leaves its safe zone within one time unit. Escape probability is important because a smaller escape probability indicates that on average the results of the query will remain unchanged for longer.

Consider the example in Fig. 20 with a range query q and the guard objects o_1, o_2, and o_3. The safe zone is shown with bold boundary. Suppose that the query q travels some distance x along a straight line in an arbitrary direction and that it crosses the boundary of the safe zone at point q'. Zhang et al. [6] presented an interesting observation for window queries which we here apply to the circular range

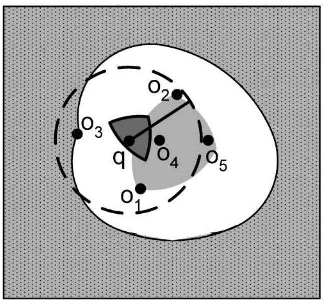


Fig. 19. Smart-update in action.

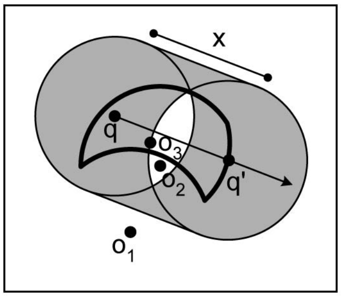


Fig. 20. Sweeping region ($x < 2r$).

queries. When a query q moves, its circle sweeps some area, which is called *sweeping region*. In Fig. 20, the shaded area corresponds to the sweeping region of the query which moved from q to q'. It is important to note that as long as the query remains in the safe zone, that is, while $x \le dist(q, q')$, the corresponding sweeping region contains no objects.

The area A of the sweeping region when the query moves a distance $x < 2r$ (as shown in Fig. 20) and a distance $x \ge 2r$ (as shown in Fig. 21) is

$$A(x) = \pi r^2 + 2rx - \begin{cases} 2r^2 arccos\Big(\dfrac{x}{2r}\Big) - x\sqrt{r^2 - \frac{x^2}{4}}, \text{ if } x < 2r \\ 0, \text{ otherwise.} \end{cases} \tag{1}$$

Since we assume uniform distribution of the objects in a unit universe, the probability p_i that an object o_i lies within the sweeping region is $A(x)$. The probability p'_i that the object o_i does not lie within the sweeping region is $(1 - A(x))$. The probability that none of the N objects lies within the sweeping region is $(1 - A(x))^N$. Hence, the probability that the query does not leave its safe zone when traveling a distance x, i.e., the probability that $x < dist(q, q')$ is $(1 - A(x))^N$. Finally, the probability that at least one of the N objects lies within the sweeping region, that is, the probability that $x \ge dist(q, q')$ is:

$$P\{x \ge dist(q, q')\} = 1 - (1 - A(x)).^N \tag{2}$$

Let the query speed v be such that the query travels distance d in one time unit. The probability of escape P_{esc} can be computed as $P\{d \ge dist(q, q')\} = 1 - (1 - A(d))^N$.

5.2 Expected Distance (m)

In this section, we analyze the expected distance m that a query travels before it leaves its safe zone. The probability

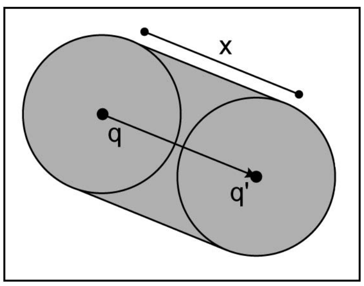


Fig. 21. Sweeping region ($x \ge 2r$).

density function $pdf(x)$ is given by the derivative of $P(x)$ presented in (2) as follows:

$$pdf(x) = 2rN(1 - A(x))^{N-1} \begin{cases} \left(1 + \sqrt{1 - (\frac{x}{2r})^2}\right), & \text{if } x < 2r \\ 1, & \text{otherwise.} \end{cases} \quad (3)$$

Integrating $x \cdot pdf(x)dx$ for x from 0 to 1 gives us the expected distance.

Unfortunately, it is difficult to integrate $x \cdot pdf(x)dx$ because the area A is represented by trigonometric functions and it makes the expression difficult to solve when $x < 2r$. We address this problem by approximating the area $A(x)$ when $x < 2r$. By plotting the equations on a graph, it can be shown that when $0 \leq x \leq 2r$, then $1.1\pi rx \leq A(x) \leq 1.3\pi rx$. We thus define the lower bound on the area as $A_{low} = 1.1\pi rx$ and the upper bound as $A_{up} = 1.3\pi rx$. We can then show that for $x < 2r$, $2rNx(1 - A_{up})^{N-1} \leq x \cdot pdf(x)dx \leq 4RNx(1 - A_{low})^{N-1}$. Thus we define the lower and upper bound on the expected distance as follows:

$$m_{up} = \int_0^{2r} 4rNx(1 - A_{low})^{N-1}dx + \int_{2r}^{1} 2rNx(1 - A(x))^{N-1}dx. \quad (4)$$

$$m_{low} = \int_0^{2r} 2rNx(1 - A_{up})^{N-1}dx + \int_{2r}^{1} 2rNx(1 - A(x))^{N-1}dx. \quad (5)$$

Exact values of m_{low} and m_{up} can be found by solving the integrals. For large values of N we have

$$m_{up} \approx \frac{0.33}{rN} \quad \text{and} \quad m_{low} \approx \frac{0.12}{rN}. \quad (6)$$

The equations for the expected distance bounds describe the relation between the expected distances, radius, and the total number of objects. More specifically, the expected distance is inversely proportional to the radius r and the number of objects N.

5.3 Expected Number of Guard Objects

We now evaluate the expected number G of guard objects. Let $d(\theta)$ be the distance a query moves in direction θ before it leaves the safe zone. Let d_{max} be the maximum of $d(\theta)$ over all θ such that $0 \leq \theta \leq 2\pi$. Let $P(x)$ be the probability that a query has $d_{max} \leq x$. We know from the theory of conditional expectation that the expected number of guard objects is given by

$$E(G) = \int_o^1 E(G|d_{max} = x)P'(x)dx, \quad (7)$$

where $E(G|d_{max} = x)$ is the expected number of guard objects for a query that has $d_{max} = x$ and $P'(x)$ is the derivative of $P(x)$ with respect to x. First, we show that $E(G|d_{max} = x) \leq 4\pi rxN$.

Consider the example of Fig. 22 where the maximum distance from q to the boundary of the safe zone is x (x corresponds to the circle shown in thick line). The circles of

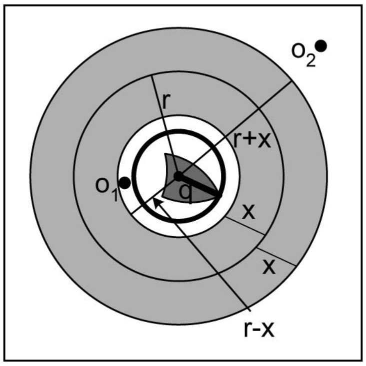


Fig. 22. Proving that $E(G|d_{max} = x) < 4\pi rxN$.

radii r, $r + x$ and $r - x$ are also shown. Any object o_i that lies in the circle of radius $r - x$ cannot be a guard object because the circle C_i of the object o_i fully contains the safe zone. This is the case because the maximum distance of o_i to the safe zone $maxdist(o_i, S) \leq dist(q, o_i) + x \leq r$. Hence, the object o_i cannot affect the shape of the safe zone.

Similarly, any object o_j that lies outside the circle of radius $r + x$ cannot affect the shape of the safe zone as the minimum distance of o_j to the safe zone $mindist(o_j, S) \geq dist(q, o_j) - x \geq r$. Fig. 22 shows two objects o_1 and o_2 and both cannot be the guard objects.

As discussed above, only those objects that have a distance from the query no less than $r - x$ and no greater than $r + x$ can be the guard objects (i.e., only the objects in the area shown shaded in Fig. 22 can be the guard objects). Thus the number G of guard objects of any query with $d_{max} \leq x$ is less than or equal to the total number of objects in the shaded area and consequently the expected number of G is less than or equal to the expected number of objects in the shaded area which is $(\pi(r + x)^2 - \pi(r - x)^2)N = 4\pi rxN$. Hence $E(G|d_{max} = x) \leq 4\pi rxN$.

For queries q for which $d_{max} \leq C \cdot m$, where C is a constant and m is the expected distance, (8) shows the upper bound of the expected number of guard objects. In other words, if we consider only the queries for which the maximum distance to the boundary of the safe zone d_{max} is not greater than $C \cdot m$, the upper bound on the expected number of guard objects is given by

$$\int_0^{C \cdot m} E(G|d_{max} = x)P'(x)dx \leq 4\pi rNCm \int_o^{C \cdot m} P'(x)dx = C \cdot 4\pi rmN. \quad (8)$$

Hence, the queries that have $d_{max} \leq C \cdot m_{up}$ have the expected number of guard objects at most:

$$4\pi rNC \times \frac{0.33}{rN} = 4.14C. \quad (9)$$

If we know C, we can obtain the upper bound on the expected number of guard objects. For instance, in our experiments, we found that 30 to 50 percent of the queries have d_{max} less than $2m_{up}$ (i.e., C is at most 2). Hence, the upper bound for such queries is 8.28.

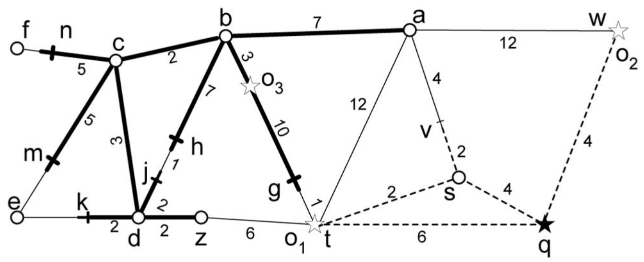


Fig. 23. Range query on a road network ($r = 10$).

6 Range Queries in Road Networks

6.1 Solution Overview

Before we outline our approach, we define a few terms.

A road network. G is a weighted graph consisting of *vertices* and *edges*. An edge between two vertices v_1 and v_2 is denoted as $e(v_1, v_2)$. Each edge has a positive weight that denotes the cost of travelling on that edge. (e.g., length of the edge, time taken to travel along the edge etc.)

Fig. 23 shows an example of a road network, together with three objects (o_1, o_2, and o_3) and a query q. For simplicity, the objects o_1, o_2 and the query q are chosen to coincide with vertices of the graph.

Segment. $seg(x, y)$ is the part of an edge between x and y where both x and y are points on the edge. By definition, an edge is also a segment defined by the end points (vertices) of the edge. Fig. 23 shows several segments including segment $seg(b, h)$ of weight seven.

Minimum network distance. $MinNetDist(a, b)$ between any two points a and b is the minimum distance from a to b (i.e., total weight of the edges and segments on the shortest path from a to b). For example, the shortest path between o_3 and v is $o_3 \rightarrow b \rightarrow a \rightarrow v$ and the $MinNetDist(o_3, v)$ is 14.

Range network of a point p (denoted as RN_p) for a given range r consists of every point of the road network G that is within the network distance r from the point p. Fig. 23 shows the range network ($r = 10$) of o_3 in thick lines.

Internal/external objects and vertices. All objects (vertices) that lie on the range network of the query q are called internal objects (vertices) and all other objects (vertices) are called external objects (vertices). Although the range network of q is not highlighted in Fig. 23, it is easy to see that the objects o_1 and o_2 are internal objects and o_3 is an external object. The vertices q, t, s, w, a are internal vertices and all other vertices are external vertices.

Safe zone is a *connected* network consisting of edges and segments such that as long as the query remains in the safe zone, its result does not change. In the example of Fig. 23, the safe zone is shown with broken lines. More specifically, the safe zone consists of $e(q, w), e(q, t), e(q, s), e(s, t)$, and $seg(s, v)$. Please note that as long as the query remains on these edges and segments the results remain the same.

The main idea of our solution is similar to our safe zone-based approach for euclidean space. More specifically, the safe zone in a road network consists of the segments of the network that are within distance r from each internal object and have distance greater than r from each external object. In other words, the safe zone is the intersection of range networks of all internal objects minus range networks of all external objects. Formally, the safe zone S is given by the expression $S = \cap_i RN_i - \cup_x RN_x$, where the intersection is taken over the range networks of all internal objects and the union is taken over the range networks of all external object.

6.1.1 Checking whether q Lies in the Safe Zone.

In contrast to the safe zone of circular range queries, the safe zone in a road network consists of edges and segments. The safe zone (e.g., the edges and segments) is sent to the query and it can easily check whether it lies in the safe zone or not.

6.2 Pruning Rules

6.2.1 Pruning Internal Objects

PRUNING RULE 6: An internal object i cannot affect the safe zone if its range network RN_i contains the whole safe zone.

Proof. Recall that the safe zone is given by $S = \cap_i RN_i - \cup_x RN_x$. If the range network of an internal object i contains the whole safe zone, it implies that the intersection of the current safe zone and the range network of i is the same as the current safe zone. Hence, the safe zone is not affected. □

Consider the example of Fig. 23 and assume that there is an object o_4 (not shown in the figure) that lies anywhere on the edge $e(q, t)$. Such an object would not affect the safe zone because its range network would cover the whole safe zone.

The above pruning rule requires computing the range networks of the internal objects in order to prune them. Next, we present a pruning rule that is less expensive.

PRUNING RULE 7: Let d_{max} be the maximum $MinNetDist(q, x)$ over every point x in the safe zone (i.e., $d_{max} = max_{x \in S}(MinNetDist(q, x))$ where S denotes the safe zone). An object o such that $MinNetDist(o, q) \leq (r - d_{max})$ cannot affect the safe zone.

Proof. We prove this by showing that the range network of any such object o contains the whole safe zone. Let x be any point in the safe zone S. The network distance between x and o satisfies $MinNetDist(o, x) \leq MinNetDist(o, q) + MinNetDist(q, x)$. Since $MinNetDist(o, q) \leq (r - d_{max})$ and $MinNetDist(q, x) \leq d_{max}$, $MinNetDist(o, x) \leq (r - d_{max}) + d_{max} \leq r$. Hence, the range network of the object o contains every point x of the safe zone. □

In the example of Fig. 23, d_{max} is six. Hence, any object that lies within distance $10 - d_{max} = 4$ of q cannot affect the safe zone. To prune an internal object, we first apply the pruning rule 7 (due to its low cost) and then apply pruning rule 6.

6.2.2 Pruning External Objects

We next present the pruning rules for external objects.

PRUNING RULE 8: An object o cannot affect the safe zone if its range network RN_o does not intersect the safe zone.

Proof. Recall that the safe zone is given by $S = \cap_i RN_i - \cup_x RN_x$. If the range network of an external object x does not intersect the safe zone, it implies that the set difference of the current safe zone and the range network of x is the same as the current safe zone. Hence, the safe zone is not affected. □

In Fig. 23, the object o_3 does not affect the safe zone because its range network does not intersect the safe zone. The next pruning rule is applicable to only the road networks where the weight of each edge corresponds to the length of the edge. For such networks, euclidean distance between any two points is always smaller than or equal to the minimum road network distance between them.

PRUNING RULE 9: An object o cannot affect the safe zone S if $mindist(o, S) \geq r$ where $mindist(o, S)$ is minimum euclidean distance of o from the safe zone S.

Proof. For any two points x and y, the euclidean distance between them is always smaller than or equal to the minimum road network distance between them. Hence, if the minimum euclidean distance between o and any point x of the safe zone is greater than r, it implies that its minimum road network distance from x is greater than r. In other words, the range network of o does not intersect the safe zone. □

PRUNING RULE 10: An external object o_j cannot affect the safe zone if $MinNetDist(o_i, o_j) \geq 2r$ where o_i is *any* internal object.

This pruning rule is similar to the pruning rule 3. The proof of correctness is basically the same except that the term range network is to be used whenever the term circle appears in the proof of the pruning rule 3.

PRUNING RULE 11: Let d_{max} be the distance as defined in the description of pruning rule 7. An object o cannot affect the safe zone if $MinNetDist(q, o) \geq r + d_{max}$.

Proof. We prove this by showing that the range network of such an object o does not intersect the safe zone. Let x be a point in the safe zone. The minimum network distance between o and x satisfies $MinNetDist(o, x) \geq MinNetDist(q, o) - MinNetDist(q, x)$. We know that $MinNetDist(q, x) \leq d_{max}$ (by definition of d_{max}) and $MinNetDist(q, o) \geq r + d_{max}$. Hence, $MinNetDist(o, x) \geq (r + d_{max}) - d_{max} \geq r$. This implies that the range network of o does not contain any point x of the safe zone (i.e., its range network does not intersect the safe zone). □

In the example of Fig. 23, d_{max} is six and an object o cannot affect the safe zone if its minimum network distance from q is at least 16.

Before we present our final pruning rule, we define a few additional terms. We say that a vertex v is a *dead* vertex if its range network RN_v does not intersect the safe zone. A path between two points a and b is called a *valid* path if the path does not contain *any* dead vertex. For example, the vertices b, c, e, and f are dead vertices because, for each of these vertices, its minimum distance to the safe zone is larger than 10. The path $d \rightarrow z \rightarrow t$ is a valid path whereas the path $d \rightarrow b \rightarrow t$ is not a valid path.

PRUNING RULE 12: An object o cannot affect safe zone if there does not exist a valid path between o and q.

Proof. By definition, the safe zone is a connected network and the query q lies on it. Moreover, it follows from the definition of dead vertex that the safe zone cannot contain any dead vertex v. This implies that if there exists a valid path between o and any point of the safe zone x then there exists a valid path between o and q. Since we know that there does not exist any valid path between o and q, this means there does not exist any valid path between o and any point x of the safe zone. This implies that there always exists a dead vertex v on every path connecting o and x. Hence, $MinNetDist(o, x) > r$ because $MinNetDist(v, x) > r$ and the path from o to x passes through v. So o cannot affect the safe zone. □

For example, in Fig. 23, any object o that lies on the edge $e(b, c)$ cannot affect the safe zone because both the vertices b and c are dead and there does not exist a valid path between o and q.

We use this pruning rule in our algorithm while exploring the road network. A vertex that is marked dead is not further explored and hence the pruning rule limits the number of explored vertices.

6.3 Algorithm

Similar to Lemma 3, it can be shown that the order in which the objects are accessed is important. More specifically, an object o_i should be accessed before an object o_j if $|r - MinNetDist(q, o_i)| < |r - MinNetDist(q, o_j)|$ (the proof is similar to the proof of Lemma 3). For this reason, we use a min-heap H that gives priority to the objects with smaller $|r - MinNetDist(q, o_i)|$ (i.e., the key of each entry e of the heap is $|r - MinNetDist(q, e)|$.

Algorithm 3 presents the details of our technique. Initially, the objects and vertices that lie in the range network of q are inserted in the min-heap H. Then, the algorithm iteratively deheaps the entries from the heap.

Algorithm 3. Network Range Query (q, r)

Input: q: the query point; r: the range
Description:
initialize a min-heap H /* key of each entry n is to bet set to $|r - MinNetDist(q, n)|$ */
insert in H vertices and objects lying on every edge that overlaps with RN_q
insert the objects lying on RN_q in answer list
while H is not empty **do**
  de-heap an entry n
  if n is a vertex **then**
    if RN_n does not intersect the safe zone **then**
      mark n as dead;
    else
      for each adjacent vertex v of n **do**
        update/insert objects lying on edge $e(n, v)$ in H
        if v is not marked dead **then**
          update/insert v in H
  else if n is an object and cannot be pruned **then**
    update the safe zone
return answer list and safe zone

If the deheaped entry n is a vertex and RN_n does not intersect the safe zone, we mark the vertex as dead. Otherwise, we process it as follows: for each of its adjacent vertices v, we insert in the heap the objects that are located on edge $e(n, v)$. It is possible that the objects on the edge $e(n, v)$ had already been inserted. In this case, for each object o lying on the edge $e(n, v)$ we update $MinNetDist(q, o)$ if its network distance from q via n is smaller than the previously stored

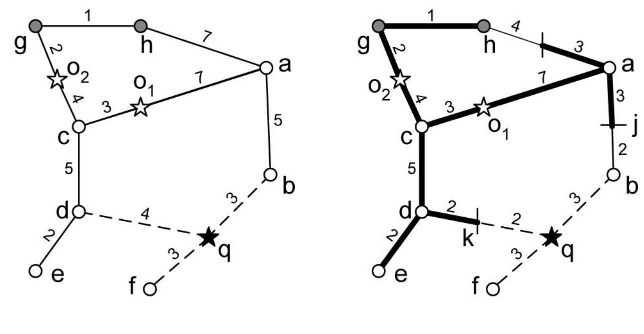


Fig. 24. Updating the safe zone ($r = 10$).

$MinNetDist(q, o)$. Its key $|r - MinNetDist(q, o)|$ is also updated accordingly. Moreover, if the vertex v is not marked dead we insert v in the min-heap. If the vertex v already exists in the heap (e.g., it was inserted when another of its neighbors was considered), we update $MinNetDist(q, v)$ and its key $|r - MinNetDist(q, v)|$ accordingly.

Finally, if the deheaped entry n is an object and it cannot be pruned by any of the pruning rules presented in the previous section, we update the safe zone using n and update d_{max}. The algorithm stops when the heap becomes empty.

6.4 Updating the Safe Zone

In this section, we present our technique for updating the safe zone (line 15 of Algorithm 3). We explain the main idea to update the safe zone for external objects. The technique for the internal objects is similar.

Recall that the safe zone is $S = \cap_i RN_i - \cup_x RN_x$ where the range network of every internal object i contributes to the intersection and the range network of every external object x contributes to the union. Hence, to update the safe zone for an external object, we need to delete the segments of the safe zone that lie within its range network.

Consider the example of Fig. 24a where the current safe zone consists of $e(q, b)$, $e(q, f)$, and $e(q, d)$ (shown in broken lines). Assume that an object o_1 is used for updating the safe zone ($r = 10$). The range network of o_1 is shown in Fig. 24b (in thick lines). The segment $seg(d, k)$ lies within the range network of o_1 and can be removed from the safe zone. Fig. 24b shows the updated safe zone which consists of $e(q, b)$, $e(q, f)$, and $seg(q, k)$. Next we show that, to update the safe zone, we do not need to compute the complete range network for every object.

We define the *vertex flow* for a vertex v with respect to an object o as $F(v, o) = r - MinNetDist(v, o)$. In Fig. 24, assume that range r is 10. The vertex flow of c with respect to o_1 is $F(c, o_1) = 10 - 3 = 7$. Similarly, $F(c, o_2) = 10 - 4 = 6$. The *maximum vertex flow* $F_{max}(v)$ of a vertex v is the maximum of $F(v, o)$ over all objects o considered so far. In Fig. 24, the maximum vertex flow of c is seven (i.e., $F_{max}(c) = 7$).

The vertex flow $F(v, o)$ denotes that every point p that lies within distance $F(v, o)$ of the vertex v lies on the range network RN_o of object o. For instance, when the range network of o_1 is computed in Fig. 24b, it discovers everything within distance $F(c, o_2) = 7$ of the vertex c.

We now show that we do not need to compute the complete range network for an object o if its range network contains a vertex v such that $F(v, o) \leq F_{max}(v)$. Consider that the range network of object o_1 has been considered and $F_{max}(v) = F(c, o_1) = 7$. When the range network of the object o_2 is being computed, the vertex c is discovered and $F(c, o_2) = 6$. Since $F(c, o_2) < F_{max}(v)$, we do not need to further explore the range network by considering the adjacent vertices of c. This is because every point within distance 7 of c has already been discovered by the range network of o_1 (whereas the range network of o_2 will discover every point within distance 6 of c).

Now, we define another condition that avoids the complete computation of range network for certain objects. If a dead vertex v is discovered during the range network computation of an object o, we do not need to further explore the vertex v. By definition of a dead vertex v, its range network does not intersect the safe zone. This means that every point within distance r of v lies outside the safe zone. Hence, the range network of o that passes through the vertex v cannot affect the safe zone. In Fig. 24, the vertices g and h are the dead vertices. When the range network of o_2 is being computed, it does not need to further explore the vertex g. Recall that the vertex c was not required to be explored because $F(c, o_2) < F_{max}(c)$. Hence, during the range network computation of o_2, only the edge $e(g, c)$ is discovered.

7 Experiments

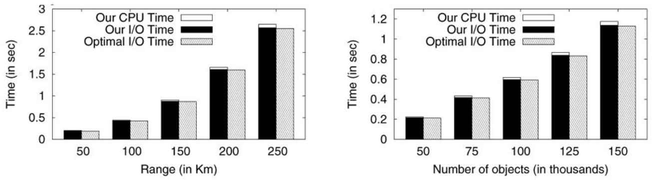
First, we present the experimental results for our approach in euclidean space. Then in Section 7.6, we present the results for range queries in road networks. To evaluate the performance of our proposed approach, we compare our approach with an optimal algorithm and a naïve algorithm. We assume that the optimal algorithm already knows the safe zone and updates the results only when the query leaves the safe zone. To compute the initial results, the optimal algorithm visits the objects that lie within the range. To update the results, the algorithm searches only the area that may contain the new answers. We only consider the I/O cost for the optimal algorithm (the CPU time is assumed to be zero).

The naïve algorithm prunes every object o_i such that its circle does not intersect with the circle of any guard object. That is, an object or rectangle can be pruned if its distance from all guard objects is greater than $2r$.

All the experiments were conducted on Intel Xeon 2.4 GHz dual CPU with 4 GBytes memory. We used real data set as well as synthetic data set. The real data set[2] contains 175,813 points of interests in North America that corresponds to a data universe of $5{,}000\ \mathrm{Km} \times 5{,}000\ \mathrm{Km}$. To verify the theoretical analysis, we created synthetic data sets consisting 50,000 to 150,000 points following uniform distribution within the same data universe size. The objects are indexed by an R-tree with node size set to $2K$.

Parameter	Range
Number of objects ($\times 1000$)	50, 75, **100**, 125, 150
Range (in Km)	50, 100, **150**, 200, 250
Average speed (in Km/hr)	40, 60, **80**, 100, 120

2. http://www.cs.fsu.edu/lifeifei/SpatialDataset.htm



(a) (b)

Fig. 25. Efficiency. (a) Radius. (b) Number of objects.

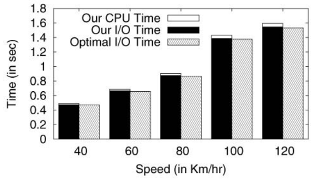


Fig. 26. Efficiency (effect of speed).

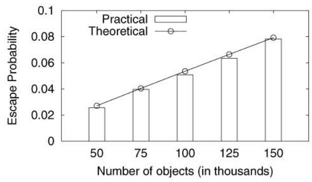
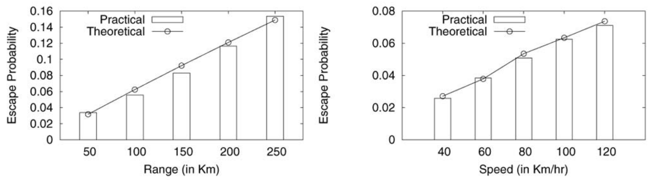


Fig. 27. Escape probability versus data cardinality.



(a) (b)

Fig. 28. Escape probability. (a) Effect of range. (b) Effect of speed.

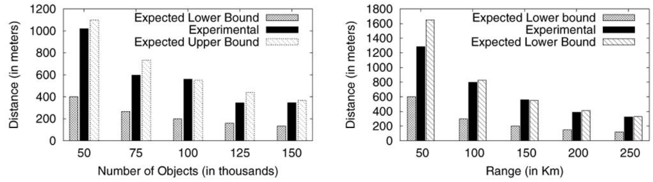
We simulated moving queries (moving cars) by using the spatio-temporal data generator described in [40]. The average speed of moving queries varies from 40 to 120 Km/hr. All queries are continuously monitored for 5 minutes and the results shown correspond to the average monitoring cost for a single query for the 5 minutes duration. All the experimental results shown correspond to the real data set except the results where we show the effect of number of objects. The table above shows the default parameters.

7.1 Cost Comparison

The cost of each algorithm consists of I/O cost (by charging 2ms for each node access) and CPU cost (assumed zero for the optimal algorithm). The naïve algorithm was at least 20 times slower[3] than our algorithm for all settings so we exclude it from figures to better illustrate the comparison of our algorithm with the optimal algorithm.

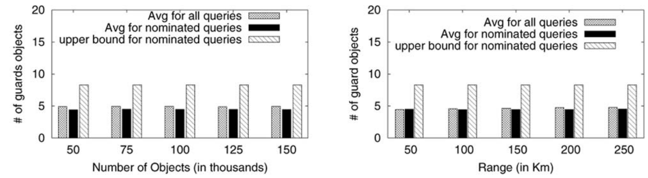
In Figs. 25 and 26, we compare the cost of our algorithm with the cost of the optimal algorithm for different ranges, different number of objects and varying speed. The performance of our algorithm is close to the optimal algorithm. The main cost for our proposed approach is the I/O cost which is very close to the I/O cost of the optimal solution. This shows that the overhead of computing the safe zone is very small compared to the cost of the range query.

3. We also compared our algorithm with naïve algorithm for in-memory data and observed 30-70 times better performance. This shows that our proposed approach performs good even for in-memory computation models.



(a) (b)

Fig. 29. Expected distance. (a) Effect of data cardinality. (b) Effect of range.



(a) (b)

Fig. 30. Number of guard objects. (a) Effect of data cardinality. (b) Effect of range.

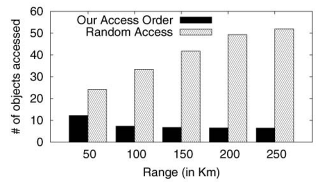


Fig. 31. Effectiveness of access order.

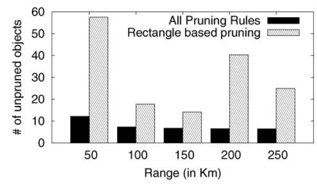
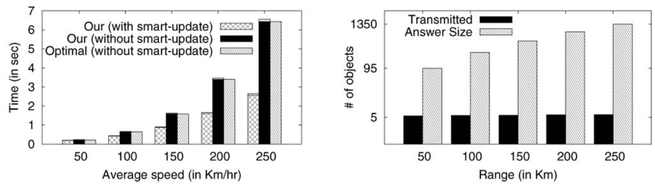


Fig. 32. Effectiveness of pruning rules.



(a) (b)

Fig. 33. Effectiveness of the smart-update. (a) Effect on cost. (b) Effect on data transmission.

7.2 Verification of the Theoretical Analysis

First, we study the escape probability and verify the theoretical results obtained. In our experiments, the escape probability of a query is computed by dividing the number of times it leaves the safe zone by the total number of movements recorded. We record the movement every second and check whether the query lies within the safe zone or not. Figs. 27 and 28 compare the escape probabilities with the theoretical results for different values of

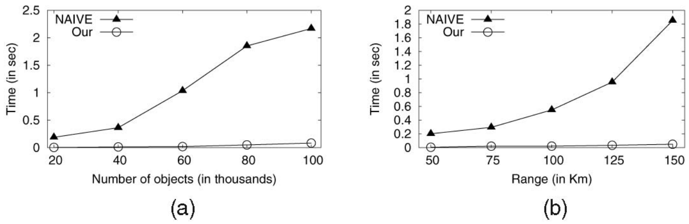


Fig. 34. Range queries in a road network. (a) Effect of data size. (b) Effect of range.

different parameters. Please note that Fig. 28 corresponds to the experiments run on the real data and it is evident that the theoretical results are accurate even on the real data.

As expected, the escape probability increases with the number of objects. The range and the speed have a similar effect on the escape probability. The results demonstrate that the escape probability is small, which shows the effectiveness of our proposed approach in real world settings.

In Fig. 29, we show the expected distance for queries run on the synthetic data set with increasing number of objects and increasing range of the query. It shows that the actual expected distance is close to the expected bounds we obtained in Section 5. Moreover, the actual expected distance is from 300 to 1,200 meters.

Fig. 30 shows the average number of guard objects for all queries and compares the theoretical bound with the actual number of guard objects. As stated in Section 5, our theoretical upper bound is valid for the queries for which maximum distance to the safe zone is smaller than $C \cdot m_{up}$ where C is a constant. We observed that when C is set to 2, 30 to 50 percent queries satisfy the constraint. We call such queries the nominated queries.

In Fig. 30, we show the average number of guard objects for all queries as well as the average number of guard objects for the nominated queries. It is interesting to note that the average number of guard objects for all queries is around five regardless of the experiment settings.

7.3 Effectiveness of the Proposed Access Order

In Fig. 31, we show the effectiveness of our proposed access order. We tried two other access orders namely *MinFirst* and *RandomAccess*. In MinFirst access order, the objects are accessed in increasing order of their distances from the query. In RandomAccess, the objects are accessed randomly. However, to improve the performance of RandomAccess, we give priority to the objects that lie within the range over the objects that lie too far from the query.

For each access order, we record the number of objects considered for updating the safe zone. MinFirst considers from 100 to 1,300 objects when the range is increased from 50-250 Km. We exclude it from Fig. 31 to better illustrate the comparison of the other two access orders. Our proposed algorithm accesses around six objects when the range becomes larger. Note that an optimal access order will access only the guard objects (the number of guard objects is around five). This shows that our proposed access order is close to the optimal access order.

7.4 Effectiveness of the Pruning Rules

In Fig. 32, we show the effectiveness of the rectangle-based pruning rules and the guard objects based pruning rules. As expected, although the rectangle-based pruning rule is computationally cheap, it is unable to prune many objects. On the other hand, the guard objects based pruning rules are more effective.

7.5 Effectiveness of Smart-Update

Fig. 33 shows the effectiveness of our proposed smart-update. In Fig. 33a, we show the cost of our algorithm with and without using the smart-update. We also show the performance of the optimal algorithm if the smart-update is not applied, i.e., every time a query leaves the safe zone, the optimal approach without the smart-update accesses all the objects within the range and sends to the client. The effectiveness of our proposed smart-update is evident from Fig. 33a. As the range increases, the performance gain by the smart-update increases because it avoids to visit a larger area.

Fig. 33b shows the average number of objects transmitted to the query whenever the server receives an update request. It also shows the total number of objects that lie within the range (shown as answer size). Please note that a log scale is used to better illustrate the trend. If the results are updated without using the smart-update, all the objects that lie within the range are to be sent again. Using our proposed smart-update approach, the number of objects that are sent to client are around five. Note that this number includes the number of guard objects that are sent to the client.

7.6 Range Queries in Road Networks

We use the road map of California[4] that consists of 21,694 road segments (edges). We generated queries moving with default speed of 80 Km/hr. Each query starts at a randomly chosen vertex. Whenever the query reaches at a vertex, one of its adjacent vertex is randomly chosen as destination and the query continues travelling. Each query is monitored for 5 minutes and the reported time is the total time for the 5 minutes duration of a query. Naïve algorithm recomputes the results whenever the query reports location update. In Fig. 34, we change the number of objects and the range of the query and observe that our approach is up to two orders of magnitude faster than the Naïve algorithm.

In Fig. 35, we show the effectiveness of our proposed access order. Similar to the experiments for euclidean distance-based queries, we observe that our access order performs better than *MinFirst* access order and the random access order. MinFirst access order was outperformed by both the random access order and our access order so we do not include it in Fig. 35.

4. http://www.cs.fsu.edu/lifeifei/SpatialDataset.htm

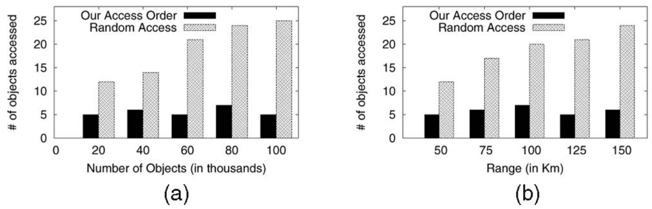


Fig. 35. Effectiveness of access order. (a) Effect of data size. (b) Effect of range.

8 Conclusion

In this paper, we presented a safe zone-based approach to efficiently monitor distance-based range queries in euclidean space and in road networks. We conducted a rigorous theoretical analysis to study the effectiveness of our safe zone-based approach for euclidean distance-based range queries. The experiment results also demonstrated that the proposed approach for euclidean distance based-range queries is close to optimal. We also showed that our network distance-based algorithm is an order of magnitude faster than a naïve approach.

References

[1] Y. Cai, K.A. Hua, and G. Cao, "Processing Range-Monitoring Queries on Heterogeneous Mobile Objects," *Proc. Conf. Mobile Data Management,* 2004.

[2] H. Hu, J. Xu, and D.L. Lee, "A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects," *Proc. ACM SIGMOD Conf.,* pp. 479-490, 2005.

[3] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring," *Proc. ACM SIGMOD Conf.,* 2005.

[4] R. Benetis, C.S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects," *Proc. Int'l Database Eng. and Applications Symp. (IDEAS),* pp. 44-53, 2002.

[5] M.A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang, "Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse knn," *Proc Conf. Very Large Database (VLDB),* vol. 2, no. 1, pp. 1138-1149, 2009.

[6] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee, "Location-Based Spatial Queries," *Proc. ACM SIGMOD Conf.,* pp. 443-454, 2003.

[7] B. Gedik and L. Liu, "Mobieyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System," *Proc. Conf. Extending Database Technology (EDBT),* 2004.

[8] Y. Tao and D. Papadias, "Time-Parameterized Queries in Spatio-Temporal Databases," *Proc. ACM SIGMOD Conf.,* pp. 334-345, 2002.

[9] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors," *Proc. Trans. Knowledge and Data Eng.,* vol. 17, pp. 1451-1464, 2005.

[10] Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proc. Conf. Very Large Databases (VLDB),* pp. 287-298, 2002.

[11] X. Xiong, M.F. Mokbel, and W.G. Aref, "Sea-Cnn: Scalable Processing of Continuous k-Nearest Neighbor Queries in Spatio-Temporal Databases," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 643-654, 2005.

[12] G.S. Iwerks, H. Samet, and K.P. Smith, "Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates," *Proc. Conf. Very Large Databases (VLDB),* 2003.

[13] X. Yu, K.Q. Pu, and N. Koudas, "Monitoring k-Nearest Neighbor Queries over Moving Objects," *Proc. Int'l Conf. Data Eng. (ICDE),* 2005.

[14] S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, and S.E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1124-1140, Oct. 2002.

[15] M.F. Mokbel, X. Xiong, and W.G. Aref, "Sina: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases," *Proc. ACM SIGMOD Conf.,* pp. 623-634, 2004.

[16] X. Wang and W. Wang, "Continuous Expansion: Efficient Processing of Continuous Range Monitoring in Mobile Environments," *Proc. Conf. Database Systems for Advanced Applications (DASFAA),* pp. 890-899, 2006.

[17] H. Wang, R. Zimmermann, and W.-S. Ku, "Distributed Continuous Range Query Processing on Moving Objects," *Proc. Database and Expert Systems Applications (DEXA),* pp. 655-665, 2006.

[18] B. Gedik, K.-L. Wu, P.S. Yu, and L. Liu, "Motion Adaptive Indexing for Moving Continual Queries over Moving Objects," *Proc. Conf. Information and Knowledge Management (CIKM),* 2004.

[19] K.-L. Wu, S.-K. Chen, and P.S. Yu, "Incremental Processing of Continual Range Queries over Moving Objects," *IEEE Trans. Knowledge Data Eng.,* vol. 18, no. 11, pp. 1560-1575, Nov. 2006.

[20] B. Zheng and D.L. Lee, "Semantic Caching in Location-Dependent Query Processing," *Proc. Symp. Spatial and Temporal Databases (SSTD),* pp. 97-116, 2001.

[21] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. Symp. Spatial and Temporal Databases (SSTD),* pp. 79-96, 2001.

[22] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The V*-Diagram: A Query-Dependent Approach to Moving Knn Queries," *Proc. Very Large Databases,* vol. 1, no. 1, pp. 1095-1106, 2008.

[23] M. Hasan, M.A. Cheema, X. Lin, and Y. Zhang, "Efficient Construction of Safe Regions for Moving knn Queries over Dynamic Datasets," *Proc. Symp. Spatial and Temporal Databases (SSTD),* pp. 373-379, 2009.

[24] C.S. Jensen, J. Kolárvr, T.B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," *Proc. Geographic Information Systems (GIS),* pp. 1-8, 2003.

[25] H.-J. Cho and C.-W. Chung, "An Efficient and Scalable Approach to cnn Queries in a Road Network," *Proc. Conf. Very Large Databases (VLDB),* pp. 865-876, 2005.

[26] M.R. Kolahdouzan and C. Shahabi, "Continuous k-Nearest Neighbor Queries in Spatial Network Databases," *Proc. Spatio-Temporal Database Management,* pp. 33-40, 2004.

[27] M. Kolahdouzan and C. Shahabi, "Voronoi-Based k Nearest Neighbor Search for Spatial Network Databases," *Proc. Conf. Very Large Databases (VLDB),* pp. 840-851, 2004.

[28] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," *Proc. Conf. Very Large Databases (VLDB),* pp. 802-813, 2003.

[29] K. Mouratidis, M.L. Yiu, D. Papadias, and N. Mamoulis, "Continuous Nearest Neighbor Monitoring in Road Networks," *Proc. Conf. Very Large Databases (VLDB),* pp. 43-54, 2006.

[30] C. Shahabi, M.R. Kolahdouzan, and M. Sharifzadeh, "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases," *Proc. ACM Symp. Geographic Information Systems (GIS),* pp. 94-10, 2002.

[31] S. Shekhar and J.S. Yoo, "Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches," *Proc. Symp. Geographic Information Systems (GIS),* pp. 9-16, 2003.

[32] D. Stojanovic, A.N. Papadopoulos, B. Predic, S. Djordjevic-Kajan, and A. Nanopoulos, "Continuous Range Monitoring of Mobile Objects in Road Networks," *Data Knowledge Eng.,* vol. 64, no. 1, pp. 77-100, 2008.

[33] F. Liu, T.T. Do, and K.A. Hua, "Dynamic Range Query in Spatial Network Environments," *Proc. Database and Expert Systems Applications (DEXA),* pp. 254-265, 2006.

[34] H. Wang and R. Zimmermann, "Snapshot Location-Based Query Processing on Moving Objects in Road Networks," *Proc. Symp. Geographic Information Systems (GIS),* p. 50, 2008.

[35] Z. Chen, H.T. Shen, X. Zhou, and J.X. Yu, "Monitoring Path Nearest Neighbor in Road Networks," *Proc. ACM SIGMOD Conf.,* pp. 591-602, 2009.

[36] H.-P. Kriegel, P. Kröger, and M. Renz, "Continuous Proximity Monitoring in Road Networks," *Proc. Symp. Geographic Information Systems (GIS),* p. 12, 2008.

[37] A. Küpper and G. Treu, "Efficient Proximity and Separation Detection Among Mobile Targets for Supporting Location-Based Community Services," *Mobile Computing and Comm. Rev.,* vol. 10, no. 3, pp. 1-12, 2006.

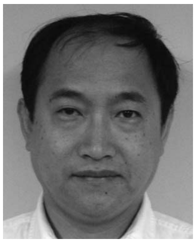
[38] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Conf.,* pp. 47-57, 1984.

[39] M.A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W.W. 0011, "Multi-Guarded Safe Zone: An Effective Technique to Monitor Moving Circular Range Queries," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 189-200, 2010.

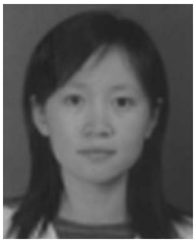
[40] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *GeoInformatica*, vol. 6, no. 2, pp. 153-180, 2002.

Muhammad Aamir Cheema received the BSc degree in electrical engineering from University of Engineering and Technology, Lahore, in 2005, and the Master of Engineering degree in computer science and engineering from the University of New South Wales, Australia, in 2007. He is currently working toward the PhD degree in the School of Computer Science and Engineering, the University of New South Wales, Australia. He also served as associate lecturer at the University of New South Wales, Australia, in 2007. He received 2009 CiSRA best research paper award and the best research paper award of Autralasian Database Conference 2010. His current research interests include spatio-temporal databases, location-based services, mobile and pervasive computing and probabilistic databases.

Ljiljana Brankovic received the bachelors degree in electrical engineering from the University in Belgrade and the PhD degree in computer science from the University of Newcastle, Australia. She is an associate professor in the School of Electrical Engineering and Computer Science, University of Newcastle and a former head of discipline of Computer Science and an assistant dean (Postgraduate) at the Faculty of Electrical Engineering and Built Environment. Her research interests include databases and data security, privacy preserving data mining, and combinatorics and graph theory. She is a fellow of the Institute of Combinatorics and its Applications, life member of the Combinatorial Mathematics Society of Australasia and a former Chair of the National Committee for Computer Security of the Australian Computer Society.



Xuemin Lin received the BSc degree in applied math from Fudan University in 1984 and the PhD degree in computer science from the University of Queensland in 1992. During 1984-1988, he studied for the PhD degree in applied math at Fudan University. Currently, he is working as a professor in the School of Computer Science and Engineering, the University of New South Wales. He has been the head of database research group at University of New South wales (UNSW) since 2002. Before joining UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. Currently, he is an associate editor of *ACM Transactions on Database Systems*. His current research interests lie in data streams, approximate query processing, spatial data analysis, and graph visualization.



Wenjie Zhang received the BS degree and MS degree in computer science from Harbin Institute of Technology, China. Currently he is working as a postdoc research fellow in School of Computer Science and Engineering, the University of New South Wales, Australia. She received the PhD degree from the University of New South Wales, Her research focuses on spatial data analysis, stream processing, and uncertain data management. She has published papers in conferences and journals including SIGMOD, VLDB, ICDE, *VLDBJ*, *TKDE*, and *Information Systems*. She is also the recipient of Best Paper Award of National Database Conference of China 2005, APWebWAIM 2009, and Australasian Database Conference 2010.

Wei Wang received the PhD degree in computer science from Hong Kong University of Science and Technology in 2004. Currently, he is a senior lecturer at the School of Computer Science and Engineering at University of New South Wales, Australia. His research interests include integration of database and information retrieval techniques, similarity search, and query processing and optimization.