Multi-Guarded Safe Zone: An Effective Technique to Monitor Moving Circular Range Queries

Muhammad Aamir Cheema[†], Ljiljana Brankovic[⋆], Xuemin Lin[†], Wenjie Zhang[†], Wei Wang[†]

[†]*The University of New South Wales, Australia*
{macheema,lxue,zhangw,weiw}@cse.unsw.edu.au

[⋆]*The University of Newcastle, Australia*
ljiljana.brankovic@newcastle.edu.au

***Abstract*— Given a positive value r, a circular range query returns the objects that lie within the distance r of the query location. In this paper, we study the circular range queries that continuously change their locations. We present an efficient and effective technique to monitor such moving range queries by utilising 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 re-evaluated unless it leaves the safe zone. The shape of the safe zone is defined by the so-called *guard objects*. The cost of checking whether a query lies in the safe zone takes k distance computations, where k is the number of the guard objects. 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) To show the effectiveness of the safe zone, we theoretically evaluate the probability that a query leaves the safe zone within one time unit and the expected distance a query moves before it leaves the safe zone. Additionally, for the queries that have diameter of the safe zone less than its expected value multiplied by a constant, we also give an upper bound on the expected number of guard objects. This upper bound turns out to be a constant, that is, it does not depend either on the radius r of the query or the density of the objects. The theoretical analysis is verified by extensive experiments. 3) Our thorough experimental study demonstrates that our proposed approach is close to optimal and is an order of magnitude faster than a naïve algorithm.**

I. 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 Euclidean distance between an object $o \in O$ and the query q. A circular range query returns every object $o \in O$ that lies within distance r of the query location q, i.e., every object such that $dist(o, q) \leq r$. We call such query a "circular range query" since the search space is a circle around the query. 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. Circular 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 circular 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 past few years [1], [2], [3], [4], [5], [6], [7].

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 certain time and only have up to 90min available for lunch. They may want to continuously monitor restaurants within 10km of their current location so that they can choose a restaurant that serves their favorite meals, and will not take more than 15 min to reach. As another example, a ship sailing through an ocean might need to continuously monitor the icebergs around it to avoid accidents.

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 1km 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 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 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.

a. The computation of the safe zone reduces the overall computation time because the query needs to be re-evaluated 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.

b. Although the shape of the safe zone may be arbitrarily complex, we can still efficiently check whether the query lies within it, as we utilize the fact that the safe zone only depends on the so-called *guard* objects. This checking only 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 5. 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.

c. We do not require the data objects to be stored in the main-memory, which allows our approach to work on the systems with limited main-memory (e.g., GPS navigation systems).

d. When an update request is received, the server computes the new safe zone and the results. 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.

e. In client-server paradigm, our proposed approach does not require 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 [5], this technique is not applicable to the moving circular range queries. In Section II we show that it is not possible to extend this technique to the case of the circular range queries as the problems of monitoring moving window queries and circular 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. In Section VI, we show that our proposed approach can handle object updates (i.e., the insertion and deletion of the objects from the underlying dataset).
- We present a rigorous theoretical analysis to verify the effectiveness of our safe zone based approach. 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.

The remainder of the paper is organized as follows. In Section II, we give an overview of the related work. We introduce our framework and pruning rules in Section III, while in Section IV we present our safe zone based solution to the moving circular range queries. Theoretical analysis is presented in Section V, and we extend our approach to handle the object updates in Section VI. The experimental results are reported in Section VII. Section VIII concludes the paper.

II. Related Work

Continuous monitoring of spatial queries has been extensively studied in recent past [8], [9], [2], [10], [3], [11], [12], [5]. Prabhakar et al. [13] proposed velocity constrained indexing and query indexing for continuous evaluation of static queries over moving objects. Mokbel et al. [14] 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 [6], [1], [15], [16]. Gedik et al. [6] 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 [17], the authors propose 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. [18] 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 [19], [20], [5], [21], [22] and moving window queries [5]. 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 et al. [7] 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 [7], the techniques to answer TP kNN queries, TP window queries and TP join queries are presented.

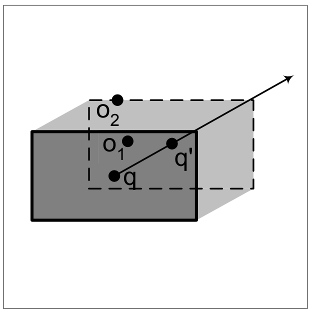


Fig. 1. A time-parameterized window query

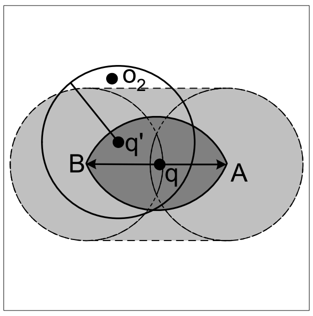


Fig. 2. TP circular queries cannot be used to construct safe zone

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. [5] 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 towards 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 [5]); 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 [5] 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 [5] is that if none of the TP queries issued towards 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 towards 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.

III. Framework

In this section, we first give a solution overview and introduce the terms and notation used in this paper. We then present a set of pruning rules used to efficiently construct the safe zone.

A. 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 centre at the location of the object o_i. Fig. 3 shows the circles for the objects o_1, o_2 and o_3.

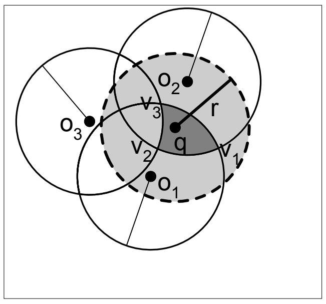


Fig. 3. A range query and its safe zone

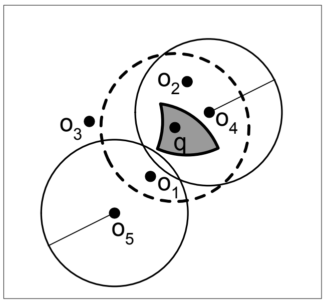


Fig. 4. Some objects do not affect the safe zone

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

[1]We use the term current safe zone because the 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.

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) Data structure at a glance: All objects are indexed by a disk-resident R-Tree [23]. 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.

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 5. 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.

B. 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.

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.

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

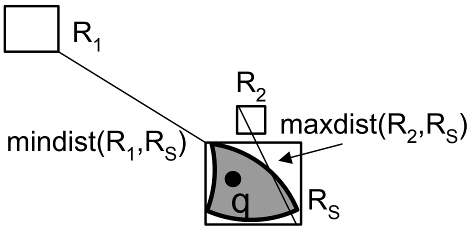


Fig. 5. Pruning using the approximation of safe zone

The proofs are straightforward and we omit them. In the example in 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. Pruning rule 1 prunes the rectangles that contain internal objects and the pruning rule 2 prunes the rectangles containing external objects.

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.

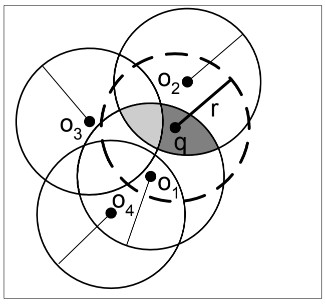


Fig. 6. Pruning rule 3

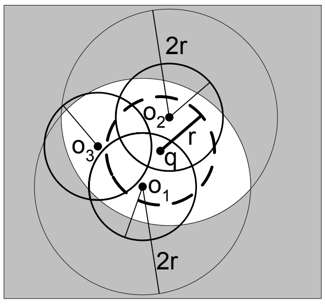


Fig. 7. Area pruned by the rule 3

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 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 centre 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 counter-clockwise. 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° 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.

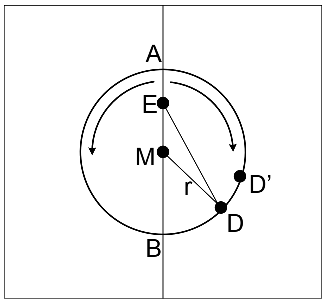


Fig. 8. Observation 1

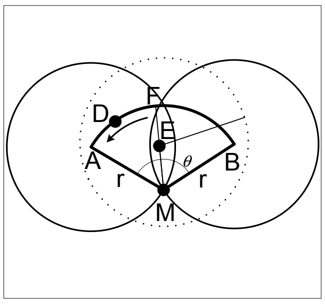


Fig. 9. Lemma 1

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

LEMMA 1 : Let $\overset{\frown}{AB}$ be an arc of radius r with subtending angle $\theta < 180°$ where A and B are the end points of the arc and M is the centre (as shown in Fig. 9). Let C_A and C_B be two circles of radius r centred 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 centre at E (the dotted circle in Fig. 9) contains every point of the arc $\overset{\frown}{AB}$.

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 $\overset{\frown}{AB}$ 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 $\overset{\frown}{AF}$ and the proof for the arc $\overset{\frown}{FB}$ is similar. By Observation 1, we know that any point D that lies on the arc $\overset{\frown}{AF}$ 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 $\theta \geq 180°$ as the line joining M and E intersects the arc $\overset{\frown}{AB}$ at point F which is the maximum distance from E to the circle and is greater than r (Fig. 10).

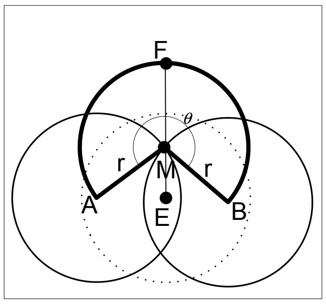


Fig. 10. When $\theta > 180°$

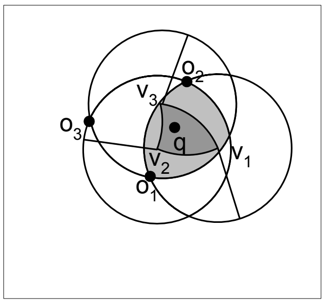


Fig. 11. Pruning rule 4

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°. 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 centred at vertices of the safe zone. From Lemma 1, we know that the circle centred 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 centres 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 $\overset{\frown}{AB}$ be an arc with centre at M, radius r and subtending angle $0 < \theta < 360°$ as shown in Fig. 12. The distance of E from every point of the arc $\overset{\frown}{AB}$ 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 $\overset{\frown}{AB}$ 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 $\overset{\frown}{AB}$. 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 $\overset{\frown}{AB}$.

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 $\overset{\frown}{AB}$ we have either $dist(E_2, D) \geq dist(E_2, A) > r$ or $dist(E_2, D) \geq dist(E_2, B) > r$. ■

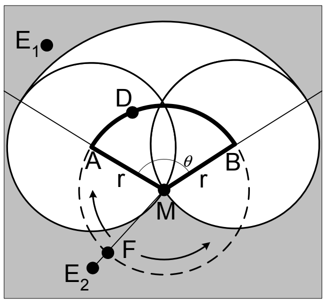


Fig. 12. Lemma 2

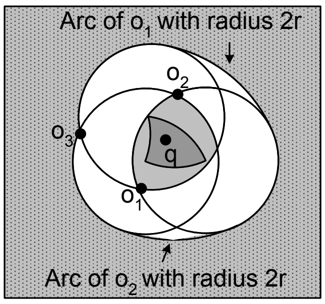


Fig. 13. Pruning by rules 4 and 5

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.

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. 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.

IV. 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. Secondly, 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.

A. 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, 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.

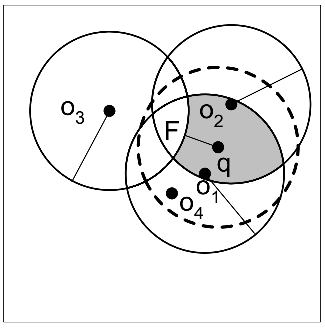


Fig. 14. Importance of the order in which objects are accessed

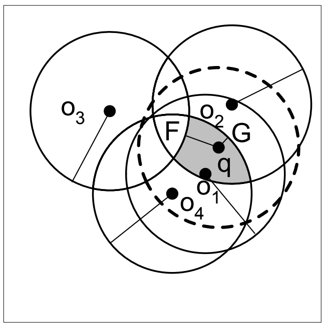


Fig. 15. o_1 is not a guard object anymore

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 $\mid dist(o_3, q) - r \mid$. Any object o_4 that lies closer to the boundary than o_3 has a point G on its circle with distance $\mid dist(o_4, q) - r \mid$ from the query, which is less than $\mid dist(o_3, q) - r \mid$ (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. 29 in Section VII). Next, we present an efficient algorithm that accesses the objects in the proposed order.

B. Algorithm

We use R-Tree [23] 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 [23].

Algorithm 1 outlines the solution. A min-heap is initialized with the root entry of the R-tree. The entries are de-heaped iteratively until the heap becomes empty. If a de-heaped 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 de-heaped 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 de-heaping 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 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).

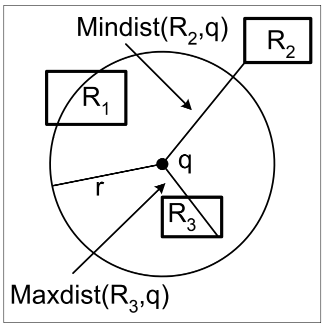


Fig. 16. Minimum distance from the boundary

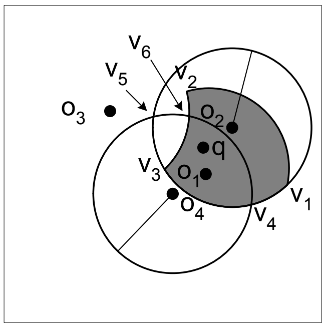


Fig. 17. Illustration of the trimming (Algorithm 2)

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.

C. 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.

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.

D. 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.

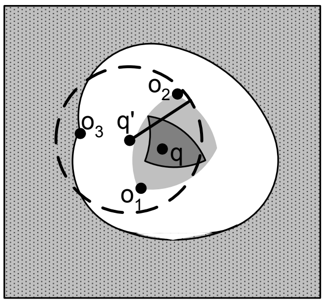


Fig. 18. q leaves the safe zone

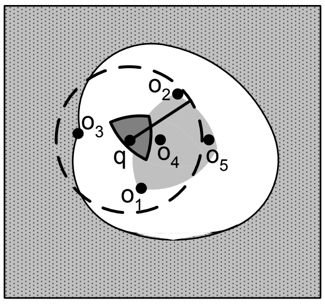


Fig. 19. Smart-update in action

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 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.

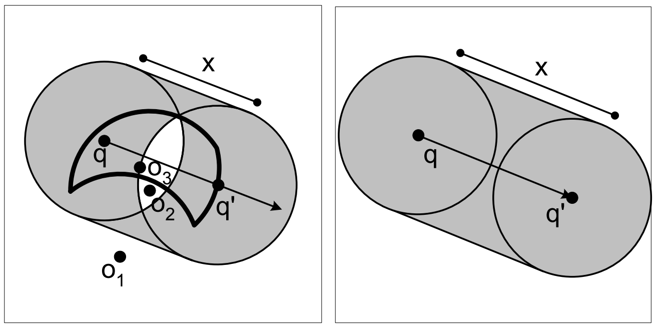
In the example in Fig. 19, the object o_4 is not sent to the query because it lies in the previous internal pruned area and the query already contains it. However, the object o_5 must be sent so that the query removes it from its answer list.

V. Theoretical Analysis

In this section we present 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.

A. Escape Probability (P_{esc})

We first analyse *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.



(a) For $x < 2r$ (b) For $x \geq 2r$

Fig. 20. Sweeping region

Consider the example in Fig. 20(a) 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 along a straight line in an arbitrary direction and crosses the boundary of the safe zone at point q'. Zhang et al. [5] presented an interesting observation for window queries which we here apply to the circular range queries. When a query q moves, its circle sweeps some area, which is called *sweeping region*. Fig. 20(a) shows 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 $x \leq dist(q, q')$, the corresponding sweeping region contains no objects.

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

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

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 $x < dist(q, q')$ 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 $x \geq dist(q, q')$ is:

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

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 $P\{d \geq dist(q, q')\} = 1 - (1 - A(d))^N$.

B. Expected distance (m)

In this section, we analyse the expected distance m that a query travels before it leaves its safe zone. The probability density function $pdf(x)$ is given by the derivative of $P(x)$ presented in Equation (2) as follows:

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

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 $x < 2r$. 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:

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

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

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

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

The equations for the expected distance bounds describe 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.

C. 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

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

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. 21 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 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 the guard object because the circle C_i of the object o_i fully contains the safe zone. This is the case because 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.

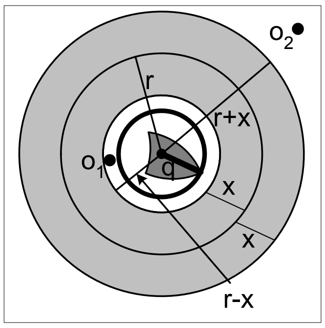


Fig. 21. Proving that $E(G|d_{max} = x) < 4\pi rxN$

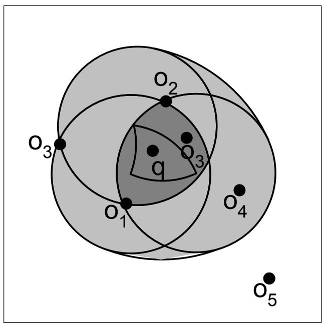


Fig. 22. Impact region shown shaded

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. 21 shows two objects o_1 and o_2 and both cannot be the guard objects[2].

As discussed above, only those objects that have 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. 21 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

[2]This observation can be used as a pruning rule. We observed that the area pruned by this pruning rule is less than the rectangle based pruning in most of the cases. On the other hand, the cost of rectangle based pruning is similar to this pruning rule. Hence, we do not present this as a pruning rule.

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, Equation (8) shows the upper bound of expected number of guard objects. In other words, if we consider only queries for which maximum distance to the boundary of 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:

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

In our experiments, we found that 30% to 50% of the queries have d_{max} less than $2m_{up}$. Hence, upper bound on expected number of guard objects for such queries is 8.28.

VI. Extension to handle object updates

Consider the example of a person driving a car who is interested in available parking spaces within 1Km. When a parking space is occupied, the parking meter notifies the server that the space has been occupied and the server treats this object (the parking space) as if it has disappeared (this object will not be considered for any query). Similarly, when a parking space becomes available, the server marks it as appeared and this should be reported to every query that contains it in its range.

For such datasets where the objects may appear or disappear, the server needs to store the query information in order to notify it if an object update affects its results or its safe zone. Below, we present an efficient approach that checks the effect of object update only for some selected queries and guarantees that the results of all other queries are unaffected.

First, we define *impact region*. The impact region of a query is the area such that any object update inside it affects the query results (and/or its safe zone) and any object outside this region does not affect the query results or the safe zone. The impact region for a query q is the area shown shaded (dark and light shaded area) in Fig. 22. This is because from pruning rule 5 we know that any object that lies outside this area cannot affect the safe zone.

We handle the object updates as follows.

Case 1. An object o_i (o_3 in Fig. 22) appears/disappears in the internal pruned area (the dark shaded area). Note that such update does not affect the safe zone. However, such object lies within the range of the query and the query must be notified. Hence, the server notifies the query that the object o_i has appeared/disappeared.

Case 2. An object o_i (o_4 in Fig. 22) appears/disappears in the light shaded area. Such object affects the safe zone. Hence, the server computes the new safe zone, updates the results and notifies the changes to the query.

All other object updates do not affect the query and can be ignored. In Fig. 22, the object o_5 does not affect the query results or its safe zone and is ignored.

To efficiently determine the queries that contain an object update in their impact regions, we use a grid structure. More specifically, each cell of the grid contains a *query list* which contains the query ids for every query that has its impact region overlapping the cell. When an object appears/disappears, we locate the cell $c_{i,j}$ relevant to its location. Only the queries that are in the query list of the cell $c_{i,j}$ might be affected. Note that the objects are not stored in the grid (we use grid only to map the location of an object to the queries it may affect).

To efficiently mark or unmark the cells that overlap with the impact region of a query, we use grid-tree [24] that is a conceptual visualization of the grid as a tree. Due to space limitations, we omit the details.

VII. Experiments

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 dataset as well as synthetic dataset. The real dataset[3] contains $175,813$ points of interests in North America that corresponds to a data universe of 5000Km$\times$5000Km. To verify the theoretical analysis, we created synthetic datasets consisting $50,000$ to $150,000$ points following uniform distribution within the same data universe size. The objects are indexed by R-tree with node size set to $2K$.

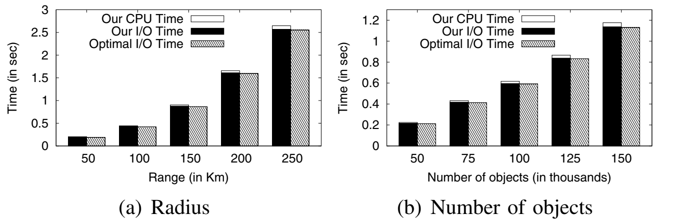
Parameter	**Range**
Number of objects ($\times 1000$)	50, 75, **100**, 125, 150
Range (in Km)	50, 100, **150**, 200, 250
Average speed (in Km/hr)	40, 60, **80**, 100, 120

We simulated moving queries (moving cars) by using the spatio-temporal data generator [25]. The average speed of moving queries varies from 40 Km/hr 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 experiment results shown correspond to the real dataset except the results where we show the effect of number of objects. The table above shows the default parameters.

[3]http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm

A. 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[4] 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 Fig. 23 and Fig. 24, we compare the cost of our algorithm with the cost of optimal algorithm for different ranges, different number of objects and varying speed. The performance of our algorithm is close to the optimal algorithm. 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.



(a) Radius (b) Number of objects
Fig. 23. Efficiency

B. Effectiveness of safe zone and verification of the theoretical analysis

First, we study the probability of escape 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. Fig. 25 and Fig. 26 compare the escape probabilities with the theoretical results for different values of different parameters. Please note that Fig. 26 corresponds to the experiments run on the real data and it is evident that the theoretical results are accurate even on the real data.

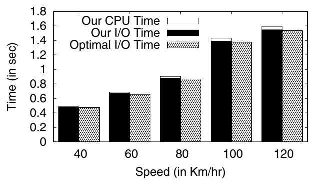


Fig. 24. Efficiency (effect of speed)

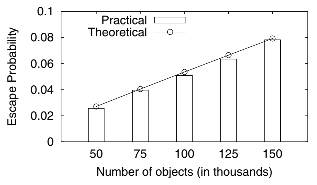
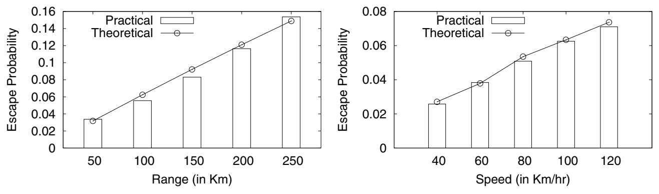


Fig. 25. Escape Probability (effec of data cardinality)

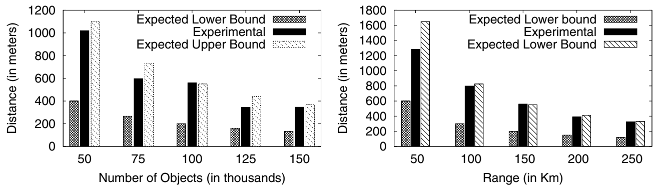
As expected, the escape probability increases with the number of objects. The range and the speed have a similar effect on the escape probability. The results demonstrate that the escape probability is small, which shows the effectiveness of our proposed approach in real world settings.

In Fig. 27, we show the expected distance for queries run on the synthetic dataset 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 V. Moreover, the actual expected distance is from 300 meters to 1200 meters.

[4]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.

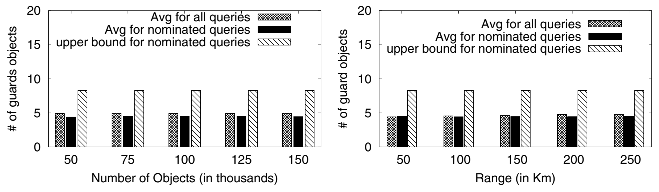


(a) Effect of range (b) Effect of speed
Fig. 26. Escape Probability



(a) Effect of data cardinality (b) Effect of range
Fig. 27. Expected Distance

Fig. 28 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 V, 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% queries satisfy the constraint. We call such queries the nominated queries.



(a) Effect of data cardinality (b) Effect of range
Fig. 28. Number of guard objects

In Fig. 28, 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 5 regardless of the experiment settings.

C. Effectiveness of the proposed access order

In Fig. 29, 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.

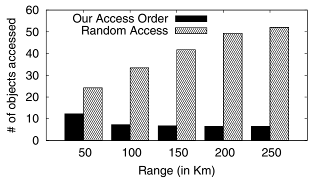


Fig. 29. Effectiveness of access order

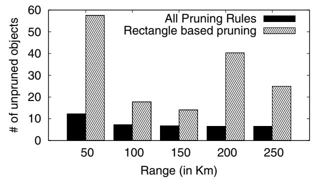


Fig. 30. Effectiveness of Pruning rules

For each access order, we record the number of objects considered for updating the safe zone. MinFirst considers from

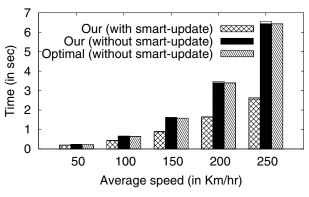
100 to 1300 objects when the range is increased from 50 Km to 250 Km. We exclude it from Fig. 29 to better illustrate the comparison of the other two access orders. Our proposed algorithm accesses around 6 objects when the range becomes larger. Note that an optimal access order will access only the guard objects (the number of guard objects is around 5). This shows that our proposed access order is close to the optimal access order.

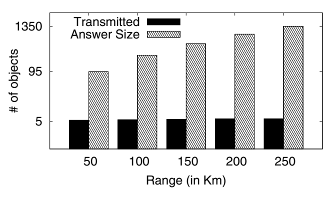
D. *Effectiveness of the pruning rules*

In Fig. 30, 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 cheap, it is unable to prune many objects. On the other hand, the guard objects based pruning rules are more effective.

E. *Effectiveness of Smart-Update*

Fig. 31 shows the effectiveness of our proposed smart-update. In Fig. 31(a), 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. 31(a). As the range increases, the performance gain by the smart-update increases because it avoids to visit a larger area.





(a) Effect on Cost (b) Effect on data transmission

Fig. 31. Effectiveness of the Smart-Update

Fig. 31(b) 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 5. Note that this number includes the number of guard objects that are sent to the client.

VIII. Conclusion

In this paper, we present a safe zone based approach to efficiently monitor moving circular range queries. We conduct rigorous theoretical analysis to study the effectiveness of our safe zone based approach. Theoretical results are verified by the extensive experimental study. The experiment results also demonstrate that the proposed approach is close to optimal and is an order of magnitude faster than a naïve approach.

Acknowledgments: This research was conducted during the second author's sabbatical at the UNSW. The third author was supported by the ARC Discovery Grants (DP0987557, DP0881035, DP0987273 and DP0666428), Google Research Award and NICTA. The fourth author was partially supported by NICTA. The fifth author's research was supported by the ARC Discovery Grants DP0987273 and DP0881779.

References

[1] Y. Cai, K. A. Hua, and G. Cao, "Processing range-monitoring queries on heterogeneous mobile objects," in *Mobile Data Management*, 2004.

[2] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *SIGMOD Conference*, 2005, pp. 479–490.

[3] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *SIGMOD*, 2005.

[4] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *IDEAS*, 2002, pp. 44–53.

[5] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD Conference*, 2003, pp. 443–454.

[6] B. Gedik and L. Liu, "Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system," in *EDBT*, 2004, pp. 67–87.

[7] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *SIGMOD Conference*, 2002, pp. 334–345.

[8] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A threshold-based algorithm for continuous monitoring of k nearest neighbors," *TKDE*, pp. 1451–1464, 2005.

[9] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *VLDB*, 2002, pp. 287–298.

[10] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *ICDE*, 2005, pp. 643–654.

[11] G. S. Iwerks, H. Samet, and K. P. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *VLDB*, 2003, pp. 512–523.

[12] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *ICDE*, 2005.

[13] 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, 2002.

[14] M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases," in *SIGMOD Conference*, 2004, pp. 623–634.

[15] X. Wang and W. Wang, "Continuous expansion: Efficient processing of continuous range monitoring in mobile environments," in *DASFAA*, 2006, pp. 890–899.

[16] H. Wang, R. Zimmermann, and W.-S. Ku, "Distributed continuous range query processing on moving objects," in *DEXA*, 2006, pp. 655–665.

[17] B. Gedik, K.-L. Wu, P. S. Yu, and L. Liu, "Motion adaptive indexing for moving continual queries over moving objects," in *CIKM*, 2004.

[18] K.-L. Wu, S.-K. Chen, and P. S. Yu, "Incremental processing of continual range queries over moving objects," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1560–1575, 2006.

[19] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing," in *SSTD*, 2001, pp. 97–116.

[20] Z. Song and N. Roussopoulos, "K-nearest neighbor search for moving query point," in *SSTD*, 2001, pp. 79–96.

[21] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The v*-diagram: a query-dependent approach to moving knn queries," *PVLDB*, vol. 1, no. 1, pp. 1095–1106, 2008.

[22] M. Hasan, M. A. Cheema, X. Lin, and Y. Zhang, "Efficient construction of safe regions for moving knn queries over dynamic datasets," in *SSTD*, 2009, pp. 373–379.

[23] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD Conference*, 1984, pp. 47–57.

[24] M. A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang, "Lazy updates: An efficient technique to continuously monitoring reverse knn," *PVLDB*, 2009.

[25] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.