MONASH University

Computing Influence in Location-based Data Sets

Arif Hidayat

A thesis submitted for the degree of Doctor of Philosophy at
Monash Univesity in 2018

Clayton School of Information Technology

Copyright Notice

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: 

Print Name: Arif Hidayat

Date: 02 March 2018

Publications during Enrolment

1. Arif Hidayat, Muhammad Aamir Cheema, David Taniar. **Relaxed Reverse Nearest Neighbors Query**. In *International Symposium on Spatial and Temporal Databases (SSTD) 2015, Hong Kong, August 26-28, 2015*

2. Arif Hidayat, Shiyu Yang, Muhammad Aamir Cheema, David Taniar. **Reverse Approximate Nearest Neighbor Queries**. in *IEEE Transactions on Knowledge and Data Engineering (TKDE), 2017*

3. Arif Hidayat, Muhammad Aamir Cheema. **QUIET ZONE: Reducing The Communication Cost of Continuous Spatial Queries**. in *ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics, 2017*

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 3 original papers published in peer reviewed journals or conferences and 1 to be submitted publications. The core theme of the thesis is influence computation in spatial databases. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology, Monash University, under the supervision of Dr. Muhammad Aamir Cheema, Dr. Campbell Wilson and Prof. Balasubramaniam Srinivasan.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapter 3,4,5 and 6 my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status *(published, in press, accepted or returned for revision, submitted)*	Nature and % of student contribution	Co-author name(s) Nature and % of Co-author's contribution	Co-author(s), Monash student Y/N
3	*Relaxed Reverse Nearest Neighbors Query*	*Published*	*80%. Problem formulation, technique, algorithm, experiment*	*1. Muhammmad Aamir Cheema, input into techniques and algorithms 15%* *2. David Taniar, input into paper 5%*	*No* *No*
4	*Reverse Approximate Nearest Neighbor Queries*	*Published*	*75%. Problem formulation, technique, algorithm, experiment*	*1. Shiyu Yang, input into some important lemmas 10%* *2. Muhammmad Aamir Cheema, input into techniques and algorithms 10%* *3. David Taniar, input into paper 5%*	*No* *No* *No*
5	*QUIET ZONE: Reducing The Communication Cost of Continuous Spatial Queries*	*Published*	*80%. Problem formulation, technique, algorithm, experiment*	*1. Muhammmad Aamir Cheema, input into techniques and algorithms 20%*	*No*
6	*Efficient Algorithm for Continuous Monitoring of Top-k Queries*		*80%. Problem formulation, technique, algorithm, experiment*	*1.Muhammmad Aamir Cheema, input into techniques and algorithms 20%*	*No*

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student signature:  Date: 02 March 2018

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor signature:  Date: 02 March 2018

Acknowledgements

First of all, I would like to thank God for keep loving me, fulfilling all my needs and guiding my life. Without His blessing, I will not be able to finish this thesis.

I am thankful to my sincere father, Muhammad Nur Ghozali, for leading me towards Him and for teaching me the meaning of life.

I am also very thankful to my supervisor Dr. Muhammad Aamir Cheema for his patience and trust in me. His encouragements have truly raised my self confidence. He has been an outstanding supervisor for me. He consistently gives a solution for every problem during my study. He always supports me whenever I need to be away from the research because of family matters.

I am thankful to Endeavour Postgraduate Scholarship, Australian Government and all staff in Scope Global Pty Ltd for the financial support and arrangement during my PhD study.

I am grateful to my supervisors Dr. Campbell Wilson and Prof. Balasubramaniam Srinivasan as well as all members in spatial database research group, Dr. David Taniar, Dr. Kiki Maulana Adhinugraha, Yathindu Rangana Hettiarachchige, Tenindra Abeywickrama, Utari Wijayanti, Anasthasia Agnes Haryanto, Zhou Shou, Ammar Sohail and Chaluka Salgado for being friendly and helpful.

The most importantly, I am thankful to my lovely wife, Yuni Idiyawati, for her support, prayer and love. I am also thankful to my parents, grand parents and all my family and friends for the help and support.

Abstract

Spatial databases have become a critical part of modern applications. Some important applications of a spatial database include Geographic Information System (GIS), Computer Aided Design(CAD), image processing and robotics. Spatial queries retrieve the required geographic data from spatial databases. In this thesis, we classify spatial queries into two categories based on their objective with regards to the notion of influence. The first category consists of the queries that aim to find the important/influential facilities. Some queries that fall into this category include range queries, k nearest neighbor queries, top-k queries and skyline queries. The second category is to find the influenced users. Queries in this category include reverse k nearest neighbor (RkNN) queries, reverse top-k queries and reverse skyline queries. In this thesis, we present efficient algorithms to solve queries in both categories. Below, we briefly describe our contributions.

We propose reverse approximate nearest neighbors (RANN) query to complement the RkNN query. This query considers relative distance between users and facilities. We propose tight and non-trivial pruning techniques to replace the existing pruning techniques which are not applicable for the RANN problem. Our extensive experimental study demonstrates that our algorithm is several orders of magnitude better than a naïve algorithm as well as a significantly improved version of the naïve algorithm.

We extend our RANN algorithm for continuous monitoring of RANN queries for the case when users are continuously moving. We propose a Voronoi based algorithm that verifies RANNs of a query using only one facility. The experiments show that our Voronoi-based algorithm is two to three orders of magnitude better than the extended state-of-the-art RkNN monitoring algorithm.

We present a safe zone based approach to efficiently monitor the result of moving top-k queries. We introduce non-trivial pruning techniques that are applicable for any monotonic scoring function. We use the techniques in our algorithm to efficiently compute the safe zone of queries. The experimental results on real data sets show that the performance of our algorithm is up to two orders of magnitude better than a naïve algorithm.

We are the first to present a generic algorithm that focuses on reducing the communication cost of a variety of continuous spatial queries. We propose a quiet zone for each object such that the moving object does not need to communicate with the server as long as it is inside its quiet zone. We show that the cost at objects is

reasonably low and our approach is feasible for devices with limited resources. The experiments show that our quiet zone approach reduces the communication cost by up to two orders of magnitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A spatial database can be defined as an optimized database to store data that defines the geographic space, such as points, lines and regions. It supports spatial data types and provides at least spatial indexing and efficient spatial join operations [7]. Some applications of spatial databases include Geographic Information Systems (GIS), image processing, Computer Aided Design (CAD), Very Large Scale Integration (VLSI) and robotics.

Spatial databases have become a critical part of modern applications. This is due to the rapid development of new technology to collect and store geographic data as well as the increased demand for analysis and utilization of this data. Many spatial queries have been introduced to retrieve the geographic objects from spatial databases. Spatial queries differ from other database queries in two aspects. First, they support geometric data types, such as points, lines and polygons. Second, these queries consider spatial relationship between the geographic objects, such as a point *inside* a polygon or a line that *intersects* another line [8]. Some spatial queries retrieve geographic objects that meet a specific requirement, such as finding closest objects to a given point or finding all objects within a given region.

In this thesis, we classify spatial queries into two categories. In the first category, the objective is to find the points in the database that are *important* for a given query point. We call these points as the influential points for the query. The notion of importance can be defined in several different ways depending on the applications. Assume that a point p is considered to be important for a point q if p is the closest point to q. In this context, a nearest neighbor query aims to retrieve an important or an influential data point, i.e., the closest data point to the query point.

In the second category, the objective of the query is to find every point p for which the query point q is an important point. Since q is an influential/important point for p, we say that p is influenced by q. In other words, the queries in this category aim to find the points that are influenced by a given query point. Queries

in this category are called reverse queries. For example, a reverse nearest neighbor query finds every data point p for which q is its nearest neighbor (i.e., closest point). Reverse queries have attracted significant research attention in the recent years due to its significance in many fields such as cluster and outlier analysis, decision support systems and location-based services [9, 10].

Although reverse queries have a broad range of applications for different types of data sets, for the ease of presentation, in this thesis we discuss these queries in the context of the data sets that consist of two types of points: facilities and users. A facility refers to a point of interest (POI) that provides some service to the users, e.g., a fuel station, a restaurant or a market. A user refers to a data point that uses the service, e.g., a driver, a diner or a shopper. Next, we present these two types of queries in detail.

1.1 Finding Important/Influential Facilities

From a user's perspective, finding the most important facilities (e.g., closest fuel stations, 3 cheapest nearby restaurants) is essential to help her selecting a facility that best matches her requirement. Consider an example of a taxi driver that is looking for nearby fuel stations. A spatial database that manages information about fuel stations can help the taxi driver to get the most important (i.e., the closest) fuel station. Some example of queries in this category include k nearest neighbors (kNN) query, range query, top-k query and skyline query. Next, we describe kNN query and details of other queries are presented in Chapter 2.

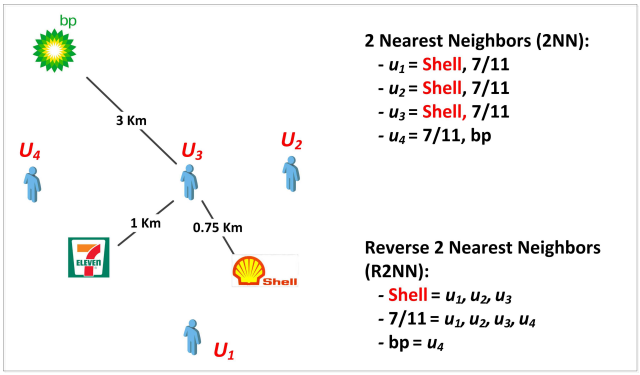


Figure 1.1: kNN and RkNN queries

Given a query point q and an integer k, a kNN query returns k facilities closest to q. In other words, a kNN query returns a set S that contains k facilities such that for each facility $f \in S$ and for any other facility $f' \notin S$, $dist(q, f) \leq dist(q, f')$, where $dist(x, y)$ denotes the distance between the locations of x and y. Consider the example of Fig. 1.1 that shows some facilities (fuel stations) and some users. The

2NNs of u_3 in this example are *Shell* and *7/11*.

1.2 Finding Influenced Users

Identifying potential users is important for businesses. Consider the example of a restaurant. Potential users of the restaurant can be used for market analysis or targeted marketing, e.g., promotion flyers may be distributed to these users because they are more likely to be influenced by deals or promotions in the flyers. Similarly, SMS or email advertisements may be sent to these user.

Influenced users can be found using reverse queries. One of the most popular reverse queries is Reverse k Nearest Neighbors (RkNN) query. The RkNN query has attracted significant research attention in the past few years. Given a query facility f and a set of users U, an RkNN query returns every user $u \in U$ which considers f as one of its k closest facilities. Consider Figure 1.1 that illustrates users looking for fuel stations. Assume that a user considers a fuel station to be important if it is one of the two closest fuel stations for her, (i.e., $k = 2$). As discussed earlier, the 2 nearest fuel stations (2NNs) of u_3 are *Shell* and *7/11*. Similarly for u_1 and u_2, they consider *Shell* and *7/11* as their 2 closest fuel stations. R2NN of *Shell* are u_1, u_2 and u_3 because these users consider *Shell* as one of their 2 closest fuel stations. u_4 is not R2NN of *Shell* because *Shell* is not one of its two closest facilities.

In the recent years, various types of reverse queries have been studied such as reverse top-k queries and reverse skyline queries. More details of these queries are provided in Section 2.

1.3 Continuous Monitoring

A *snapshot* query is a query that requires the result to be computed only once considering the current snapshot of the data. For example, a user may issue a snapshot 2NN query to find its two closest fuel stations considering her current location. She may then decide to visit or not visit a particular fuel station. In contrast to a snapshot query, a *continuous* query assumes that the underlying data is continuously changing and requires the results to be continuously updated with the changes in the underlying data. For example, in many real world applications, facilities and/or users may be continuously moving. A driver may issue a continuous 2NN query to continuously monitor her 2 closest facilities as she travels across her route.

Most of the studies for continuous monitoring of spatial queries use a *client-server* model. In this model, the moving clients send their locations to the server after every time unit and the server computes the query result accordingly and sends them back

to the querying user. Two major challenges for continuous monitoring are as follows.

- *Reducing computation cost*. The query results may need to be recomputed whenever a user or a facility changes its location. This may result in a huge computation cost.

- *Reducing the communication cost*. To maintain the correctness of the query results, a user or a facility is required to report its location to the server every time it changes its location. Since there may be a lot of users or facilities in the system, the communication cost that is triggered by these location updates may increase significantly.

1.4 Research Questions

Below, we describe research questions (RQ) that are investigated in this thesis.

- *RQ1 - How to better capture and efficiently compute the influence of facilities by considering the relative distance between users and facilities ?*
  This work was motivated by our observation that the RkNN query may fail to retrieve the potential users of a facility. An RkNN query considers relative ordering of the facilities based on their distances to the users and ignores their actual distances from the users. We show that the RkNN query may be unable to retrieve the users that are actually influenced by the query point. To better capture the notion of influence, we propose reverse approximate nearest neighbor (RANN) query. More details are presented in Chapter 3.

- *RQ2 - How to efficiently monitor continuous RANN queries?*
  In this work, we aim at continuously monitoring RANN queries in the case where users (e.g., taxi drivers) are continuously moving and facilities (e.g., fuel stations) do not change their locations.

- *RQ3 - How to continuously monitor k most influential facilities for a user?*
  Given a moving user, in this work, we continuously monitor k most important/influential facilities for the user where the importance of each facility is computed using a monotonic scoring function defined by the user.

- *RQ4 - How to reduce the communication cost for continuous monitoring of concurrent various types of spatial queries?*
  Most of the existing algorithms for continuous spatial queries focus on reducing the computation cost. Since the communication cost may also be very high, it is important to propose an algorithm that focuses on reducing the communication cost for continuous monitoring of spatial queries.

1.5 Contributions

In this section, we summarize our contributions in this thesis. For each of the above-mentioned research questions, we describe our contributions below.

1.5.1 Snapshot Reverse Approximate Nearest Neighbors (RANN) Queries

We propose a new definition of influence and introduce RANN queries that better capture the notion of influence. We show that the existing pruning techniques cannot be applied or extended for RANN queries. Based on several non-trivial observations, we propose a tight pruning technique and utilize it in our efficient algorithm to solve the snapshot RANN queries. We design a non-trivial technique as our competitor. Our extensive experiments show that our algorithm is several orders of magnitude better than the competitor.

This research [11] was published in *International Symposium on Spatial and Temporal Databases* **(SSTD)** 2015.

1.5.2 Continuous Monitoring of RANN Queries

In this study, we extend our work on snapshot RANN queries for the continuous RANN queries. We propose a Voronoi-based algorithm to efficiently monitor concurrent RANN queries. An alternative algorithm for snapshot RANN queries was also presented as a by-product of our Voronoi-based technique. We extended the state-of-the-art algorithm for continuous RkNN queries to handle continuous RANN queries and compare it with our algorithm. We conduct extensive experiments to demonstrate that our Voronoi-based algorithm significantly outperforms the state-of-the-art algorithm.

This research [12] was published in *IEEE Transactions on Knowledge and Data Engineering* **(TKDE)** 2017.

1.5.3 Continuous Monitoring of Top-k Queries

We propose a *safe zone* based approach to efficiently monitor the result of moving top-k queries. Safe zone is an area such that the results remain unchanged (and thus are not required to be recomputed) as long as the query remains inside it. We develop pruning techniques to efficiently construct the safe zone of the queries. We conduct extensive experiments on real data set to evaluate the effectiveness of our approach. The experiment result shows that our algorithm is up to two orders of magnitude better than a naïve approach. This research will be soon submitted to *The International Journal on Very Large Databases* **(VLDBJ)**.

1.5.4 Generic Framework for Continuous Monitoring of Spatial Queries

We present a generic framework to efficiently monitor the results of spatial queries. To the best of our knowledge, we are the first to present a generic framework that focuses on reducing the communication cost for many different types of spatial queries, including range queries, window queries, reverse nearest neighbor queries and reverse approximate nearest neighbor queries. Our extensive experiments on real data set shows that our algorithm is one order of magnitude better in terms of communication cost than the traditional *client-server* approach.

This research [13] was published in *International Workshop on Smart Cities and Urban Analytics* **(urbanGIS)** at *ACM SIGSPATIAL* 2017.

1.6 Thesis Organization

Below, we present the structure of the rest of this thesis.

- Chapter 2 presents a review on the related work.
- Chapter 3 describes our work on snapshot reverse approximate nearest neighbor queries.
- Chapter 4 covers our Voronoi-based technique and algorithm for continuous monitoring of reverse approximate nearest neighbor queries.
- Chapter 5 presents our research on continuous monitoring of top-k queries.
- Chapter 6 describes our technique, named *Quiet Zone*, to reduce the communication cost of continuous spatial queries.
- Chapter 7 concludes our research and describes several possible directions for future work.

Chapter 2

Literature Review

This chapter provides background on spatial queries that are related to our study. In Section 2.1, we describe the related work on queries to find the influential facilities. Section 2.2 presents the works to find the influenced users. Section 2.3 provides an overview of studies on continuous spatial queries.

2.1 Finding Influential Facilities

2.1.1 Range Query

Given a query point q and a positive value r, a range query retrieves all objects that lie within distance r from q. Formally, a range query returns every object o for which $dist(q, o) \leq r$, where $dist(q, o)$ represents the distance between q and o. Range queries have been studied in various environments, such as in Euclidean space, where Eucliden distance is used to represent the distances between users and facilities [1, 14, 15, 16, 17, 18] and road networks, where distances are computed from the shortest path between users and facilities [19, 1, 20, 21, 22, 23]. It also has been studied in different types of queries, such as snapshot queries, where the results are computed only once considering the snapshot location of the users and facilities [1, 15, 17] and continuous queries, where the results are continuously updated due to the changes in underlying data sets [24, 25, 26, 27]. Most of the works on snapshot range query in Euclidean space use R-tree [28] to index the spatial objects. R-tree groups the spatial objects based on their closeness and binds them in a minimum rectangle. The grouping continues until the top level which consists of a single root.

Fig. 2.1 illustrates the use of R-tree to answer a range query q. Firstly, the root is retrieved and the entries that overlap the range (e.g., E_1, E_2) are recursively expanded since they may contain the query results. Entries that do not overlap the

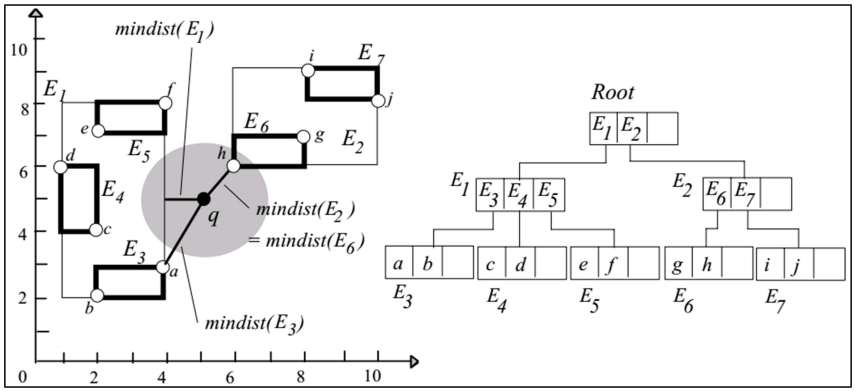


Figure 2.1: Range query with R-tree [1]

range (e.g., E_4, E_5) are skipped. If the overlapped entry is a point (e.g., h), it is returned as the query result. The search terminates after all entries are processed [1].

2.1.2 kNN Query

As mentioned in Section 1.1, a kNN query is to find the k closest objects to the query point. kNN queries have been studied in Euclidean space, where the distance is measured as the straight line distance between two objects [29, 30, 31], in road networks, where distance is the length of the shortest path between two objects' locations [32, 33, 34, 35, 36, 37, 38], in indoor space, where distance measurement considers indoor entities, such as rooms and doors, which enable and constrain indoor movement [39, 40, 41, 42] and in obstructed space, where distance is measured as the shortest path connecting two objects without crossing any obstacle [43, 44, 45, 46]. It also has been studied in various settings, such as queries with uncertain data, where due to various factors, such as delay, loss, or limitation of equipment, the exact data about objects are not always available [47, 48, 49, 50], in terrains environment, where the elevation information is taken into consideration [51, 52, 53, 54], in snapshot queries, where the results are computed based on the snapshot locations of the objects [55, 29, 56, 57, 58, 31, 59, 60] and in continuous queries, where the query result are continuously monitored [61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71].

Most of the algorithms in snapshot kNN queries use branch and bound paradigm on R-tree. These works follow either *depth-first* (DF) [29] or *best-first* (BF) [30] traversal to retrieve the kNN from R-tree. To solve a NN query, DF starts with the root and recursively visits the entries based on the minimum distance to the query until a point which is a potential NN is found. Then, DF performs backtracking to the upper level where it only visits entries whose distance to the query is smaller than the distance between query and the current NN candidate. In BF, the algorithm maintains a priority queue PQ which stores R-tree entries sorted based on the

minimum distance to the query. BF recursively visit entries e in PQ, if e is an intermediate node, its child entries are inserted into PQ. If e is a point, it is reported as an actual NN of the query. Both algorithms can be easily extended for $k > 1$.

2.1.3 Top-k Query

Unlike range and kNN queries that only consider distances between users and facilities, a top-k query considers multiple attributes of the facility, where one of the attributes is the distance to the query point. Formally, a top-k query retrieves k facilities that have the best scores. The score of each facility is computed using a user's defined scoring function w. Preference function of a user usually is a linear function with a weight assigned to each attribute that shows the importance of the corresponding attribute for the user.

Consider a user (u) that is looking for the most important fuel stations in Fig. 2.2. The user's scoring function is $w = (0.6 * price) + (0.4 * distance)$ as shown in the figure and assume that the lower score is more preferred. Based on w, the score of f_1 (resp. f_2, f_3, f_4) is $3, 4$ (resp. 2.6, 4.6, 3.8). The top-2 fuel stations for u are f_2 and f_1.

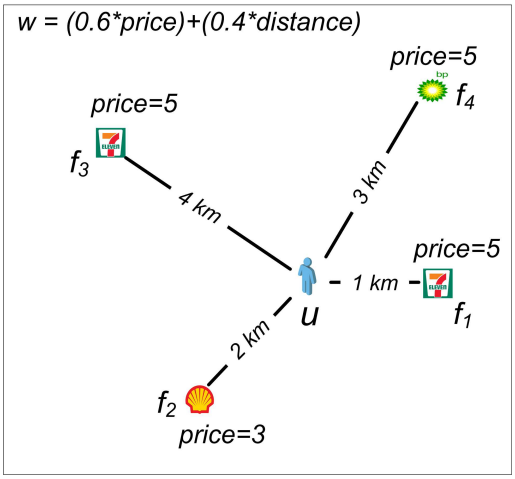


Figure 2.2: Top-k and skyline queries

Top-k queries have been extensively studied under various query models [72, 73, 74, 75, 76, 77], such as top-k selection query that report k best tuples, top-k join query that combines a set of relations based on a join condition and top-k aggregate that reports k best group of tuples. It has been studied with uncertain data, where exact data are not always available [78, 79, 80, 81]. It also has been studied in different scoring functions [82, 83], such as monotonic or generic scoring functions. Top-k spatial queries, where one of the considered attributes is the distance between users and facilities, also have attracted significant research attentions in

recent years [84, 85, 86, 87, 88, 89, 90, 91, 92] . Most recent works also consider textual relevance of the objects (top-k spatial keyword queries), where integrating inverted index on top of R*-tree is shown as the best indexing scheme [88].

2.1.4 Skyline Query

Similar to top-k query, in a skyline query, multiple attributes of the facilities are considered by the user to select a service provider. Differently, a skyline query does not require the user to specify her preference function. Consider the example of a user u looking for a fuel station in Fig. 2.2. The user may be unable to define a suitable scoring function due to various reasons such as lack of domain knowledge and incompatible attributes (e.g., km vs dollars). A skyline query tries to address this issue.

In formal definition, a skyline query retrieves all facilities that are not *dominated* by any other facility. Given a user u, a facility f dominates another facility f' if f is preferable for u than f' in at least one attribute and is not worse than f' on other attributes. In the example of Fig. 2.2, assume that u considers price and distance to choose the fuel station, we say that f_2 dominates f_3 because f_2 is better in price and distance to u than f_3. In this case, u will not likely to choose f_3 as the preferred fuel station. In Fig. 2.2, f_1 is not dominated by f_2 because even though f_1 is more expensive than f_2, it is closer to u. *Skyline* consists of every facility that is not dominated by any other facility [93] of u. In this example, the skyline of u contains f_1 and f_2.

Skyline query was firstly introduced in [93]. It has been studied in various environments, such as in Euclidean that uses the Euclidean distance [94, 95, 96, 97] and in road networks that use network distance [98, 99, 100, 101]. It also have been studied in various types of query, such as in snapshot queries, where the result are computed and reported once [93, 95, 102, 94] and in continuous queries, where the results are continuously updated according to updates in the underlying data sets [103, 104, 105, 106]. It also has been studied in uncertain data [107, 108, 109, 110], where due to some factors, the exact data of the objects are not available.

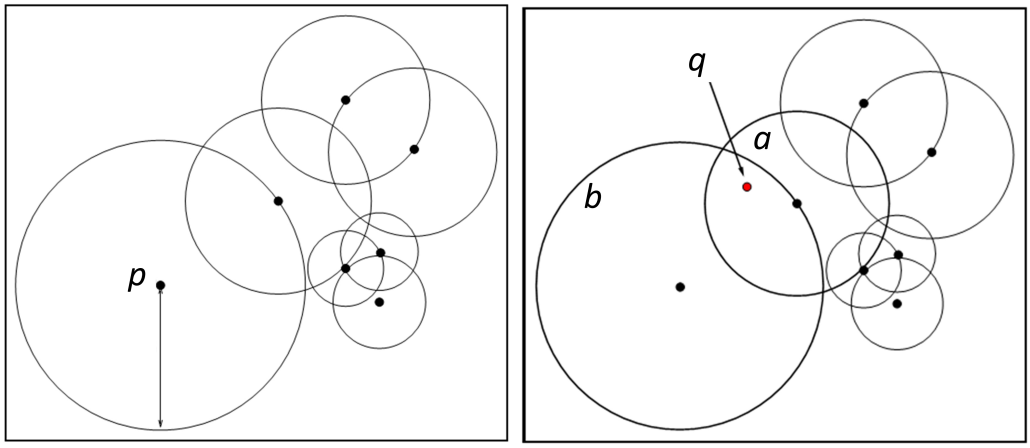
2.2 Finding Influenced Users

2.2.1 Reverse Nearest Neighbors (RNN) Query

Reverse Nearest Neighbors (RNN) query is a variant of nearest neighbors query. Given a D-dimensional data set S and a query point q, Reverse k Nearest Neighbors (RkNN) returns every user u in S which considers q as one of its k closest objects. It has attracted considerable attention ever since it was introduced in [10], where

it was proposed to be useful in various purposes, such as decision support system, profile-based marketing, referral system and document repositories maintenance.

In [10], RNN query is answered by pre-computing a circle for each data object p with p as the centre and the distance of p to its nearest neighbor be the radius of the circle. With this computation, the nearest object of p is included inside the circle. Then, for any query q, all circles containing q are recalled and their centres are returned as the query answer. Fig. 2.3 shows the steps when nearest neighbor for each object is pre-computed and then used to get centre of circles a and b as the RNN of query object q. Improvement was proposed in [111] but it still relies on pre-computation which cannot handle updates in dynamic environment efficiently.



(a) Step 1: Pre-compute Nearest Neighbor (b) Step 2: Find RNN

Figure 2.3: First RNN Computation Method

Several most significant algorithms have been introduced without using pre-computation to solve RkNN query, such as Six-Region [112], TPL [113], FINCH [114], InfZone [5] and SLICE [6]. Many variations to static RNN also have been studied, such as continuous RkNN query, where the results are continuously monitored [115, 116, 117, 118], probabilistic RkNN query, that handles unavailability of exact values of the objects [119, 120, 121, 122], RkNN query in road networks, where the distance is mesured as the distance of the shortest path that connects two objects [123, 124, 125] and reverse approximate NN query, that considers the relative distance betwenn objects [126, 12]. Since our works are on snapshot and continuous RkNN queries in Euclidean space, we will be focusing to provide previous significant works related to those area.

Six-region introduced by Stanoi et, al [112] is the first method which not relies on pre-computation. SAA algorithm in Six-region method relies on interesting properties, (i) for 2-Dimensional data set, RNN of query point q is never be more than six and (ii) for a q and a region S_i, NN(q)=RNN(q) or there is no RNN(q) in

S_i. Consider object O_1 in region S_0 in Fig. 2.4. It is the nearest neighbor of q but not the RNN of q, since there exists object O_0 which is closer to O_1. Then there is no RNN of q in region S_0.

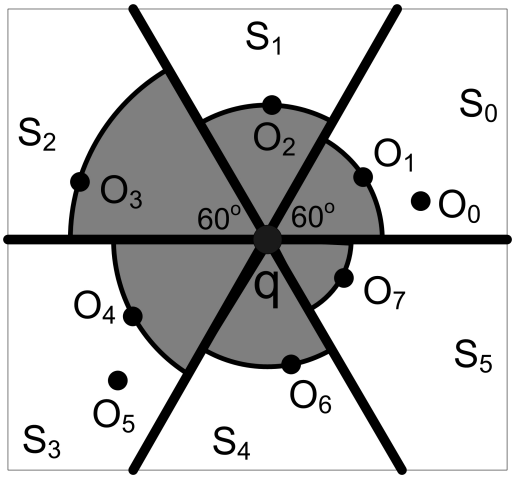
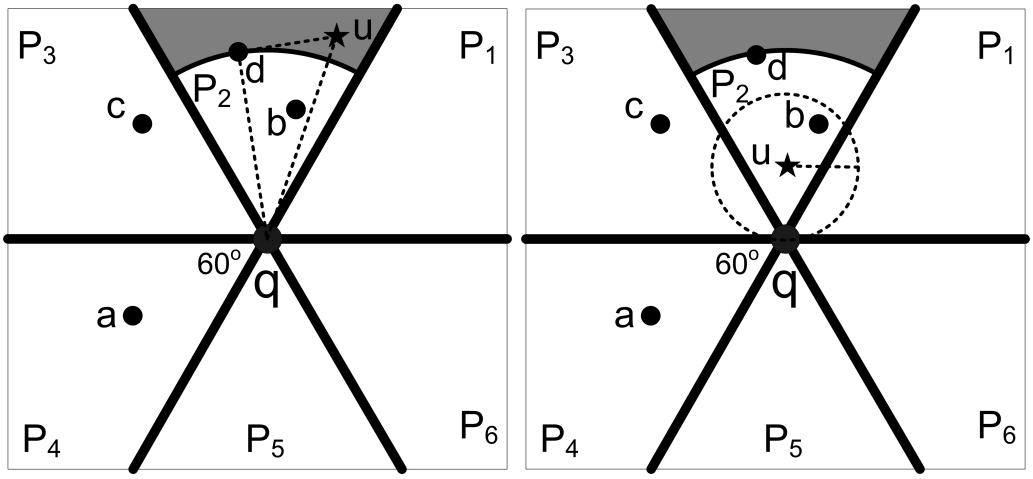


Figure 2.4: Property of SAA [2]

Based on those properties, SAA applies two steps, filtering and verification to answer RNN query. In filtering, SAA utilizes the k-nearest facility of q in each region to define the pruning area. Any user u lies inside this area has distance to q more than its distance to the k-nearest facility of q and therefore cannot be the answer of RkNN query. Consider Fig. 2.5(a) and assume $k = 2$, d is the second closest facility object to q, therefore an arc centered at q with radius $dist(d, q)$ defines the pruning area in region P_2. Any user object u lies in shaded pruning area has $dist(u, q) > dist(d, q)$, hence cannot be R2NN answer of q.

Defined pruning areas are then used to filter users and those which are not filtered become candidates of the query answer. In the following verification phase, these candidates are retrieved and verified using boolean range query. The query uses a circle centered at the candidate point u with $radius = dist(u, q)$ and immediately return true if k facility objects are found inside this circle. Consider a candidate point u in Fig. 2.5(b), the boolean range query issued from u will traverse facility R-tree to check whether k facilities objects are lie inside its circle. u is the query answer of R2NN(q) since there is only one facility object (b) found in it.

TPL [4] was the first to use half-space pruning to solve RkNN query. The half-space is created using perpendicular bisector between q and a facility point. Let AB be the line connecting two points A and B, and M be the middle point of AB, the perpendicular bisector $\perp (A, B)$ is perpendicular to line AB and passes through M. The use of half-space pruning which significantly reduces the search space makes TPL to be very popular and is adopted in many subsequent works such as in [122],

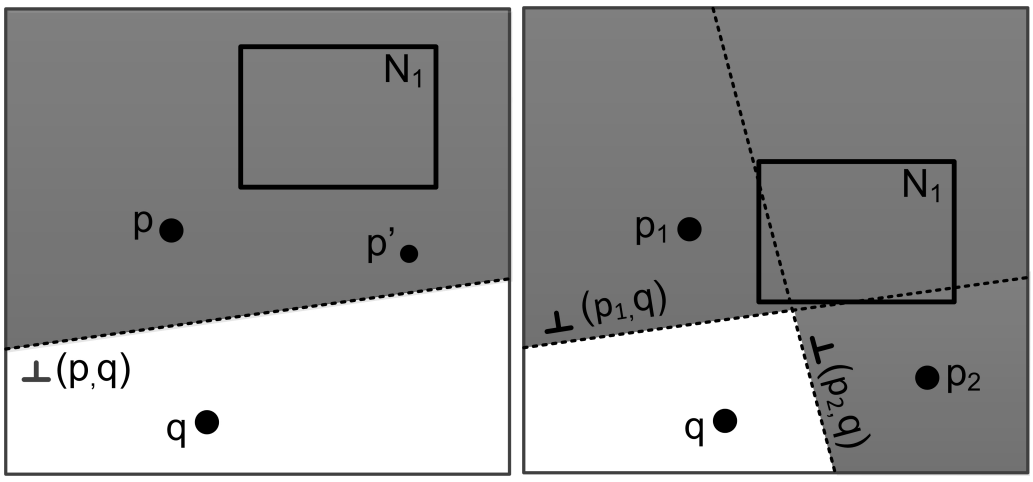


(a) Step 1: Filtering (b) Step 2: Verification

Figure 2.5: Six-region Method [3]

[5] and [114].

The perpendicular bisector of q and any object p divides the space into two half-planes, $PL(p)$ which contains p and $PL(q)$ which contains q. Consider an example in Fig. 2.6(a), any point in $PL(p)$ cannot be RNN of q as its distance to p is smaller than to q. Similarly for MBR N_1, it can be pruned if it is completely inside $PL(p)$ as it cannot contain RNN of q. Different case is shown in Fig. 2.6(b), in this example, even though MBR N_1 is not entirely fall inside one half-plane, $PL(p_1)$ or $PL(p_2)$, it is still can be pruned as it lies inside the union of $PL(p_1)$ and $PL(p_2)$.



(a) Pruning with 1 point (b) Pruning with 2 points

Figure 2.6: TPL Pruning [4]

In searching RNNs of q, TPL uses two steps called filtering and refinement. It indexes objects with R-tree and traverses it in Best-First Manner while maintaining

a min-heap containing entries of the R-tree. Initially, TPL algorithm visits the root of R-tree, open and insert its children to the heap. Entries in the heap which are pruned according to the half-space pruning rule are inserted into a refinement set S_{rfn} and those which cannot be pruned are inserted into a candidate set S_{cnd}. Consider example in Fig. 2.7, after the root and MBR N_10 are opened, entries in the heap are $N_3, N_11, N_2, N_1, N_12$. p_1 is then accessed and inserted into S_{cnd} as $dist(p_1, q) < dist(N_11, q)$ whereas p_3 is inserted into S_{rfn} since it lies in half-plane $PL(p_1)$ and will be used to verify entries in candidate set in the refinement phase.

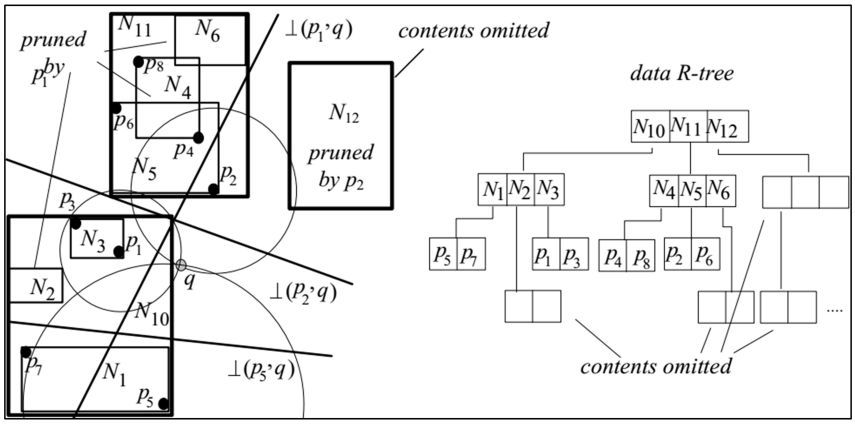


Figure 2.7: TPL Algorithm [4]

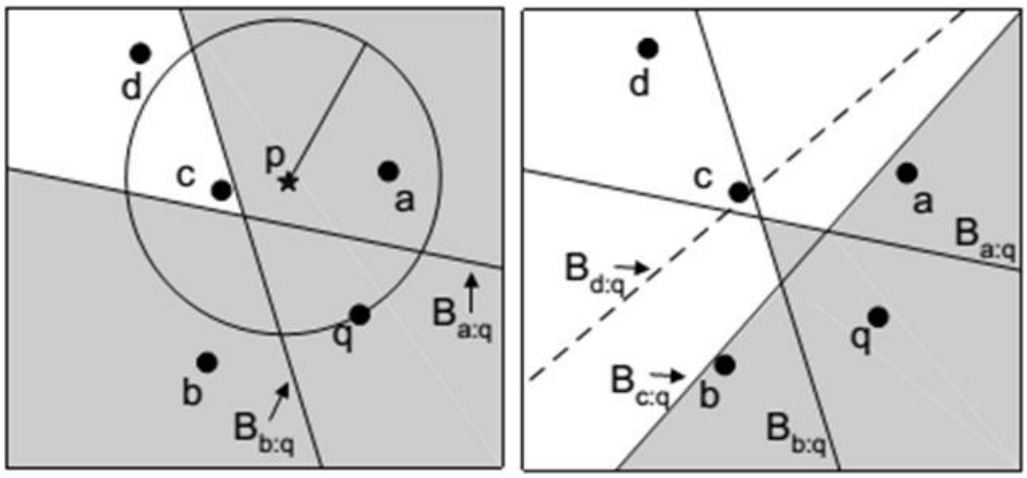
TPL algorithm keeps points and nodes encountered in filtering phase in sets of P_{rfn} and N_{rfn} to verify entries in candidate set. In this way, the algorithm only traverses the R-tree once and does not issue boolean range query in verifying the candidates. Verification is carried out in rounds, where in each round, a candidate point is verified with entries in P_{rfn} or N_{rfn}. A candidate p is discarded if there is a point $p' \in P_{rfn}$ such that $dist(p, p') < dist(p, q)$ or there is a MBR N such that $minmaxdist(p, N) < dist(p, q)$.

FINCH [114] use half-space pruning as in TPL to define pruning region. However, instead of using subsets of facility objects to filter the entries, it uses a convex polygon covering the unfiltered region. Any object inside this polygon can be filtered. FINCH algorithm approximates a convex polygon to avoid such expensive subset filtering as used in TPL. All objects inside this polygon are candidate of R2NN of q. These objects are then verified using boolean range query in the subsequent phase.

INFZONE (influence zone) [5] was introduced to improve the verification in RkNN query. Influence zone is an area for which every object inside is guaranteed to be RkNN of q. Using this zone, RkNN of a query point is obtained by calculating the zone and locating users inside it.

Even though it is also computed using half-space approach, Influence zone is different from unpruned area defined in previous algorithms utilizing half-space

method. Consider an example in Fig. 2.8 where five facility points are shown (a, b, c, d, q) and half-space pruning is applied with $k = 2$. In Fig. 2.8(a), perpendicular bisector between q and two points a and b are drawn ($B_{a:q}$ and $B_{b:q}$). Points c and d are pruned as they are in two half-planes. The unpruned area is shown shaded. In this area, a point can not be guaranteed to be RkNN of q as we still have to consider all other bisectors. As an example is point p, even though it lies in unpruned area, it is not R2NN of q since there are two facility points a and c which are closer to p. On the other hand, the shaded area in Fig. 2.8(b) which is constructed using perpendicular bisectors of all facilities is influence zone. After bisectors ($B_{c:q}$ and $B_{c:q}$) are also considered, every point lies in unpruned area is guaranteed to be R2NN of q.



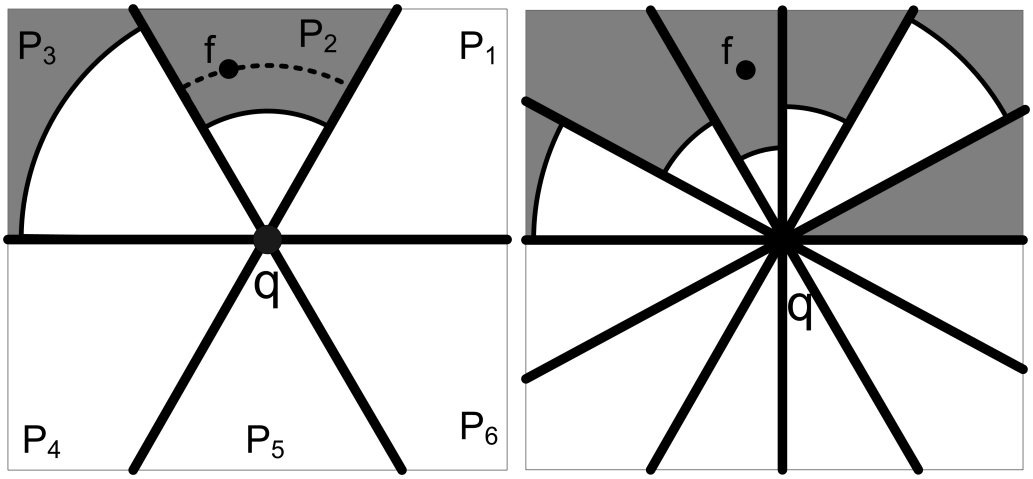
(a) Unpruned area (not influence zone) (b) Influence zone (influence zone)

Figure 2.8: Unpruned area and influence zone ($k = 2$) [5]

Influence zone such above example is computed based on naïve approach of using half-space of all facilities. However, the author proposed set of rules to significantly reduce the number of facilities which should be considered in computing the zone. Initially the influence zone is set to be the whole data space, each time a facility point is encountered, its half-plane is used to update the zone by removing space which are pruned by k facilities. An entry e is not considered in computing influence zone if for every vertex v of the most current influence zone, its minimum distance to v is greater than distance of v to q, or $mindist(e, v) > dist(v, q)$.

SLICE [6] applies filtering strategy used in six-region approach and bring back region based pruning to be an eminent approach for solving RkNN query. Unlike six-region which always has 6 partitions, SLICE divides data space in arbitrary number of partitions. Pruning strategy in SLICE is more powerful than that in six-region. This is because in SLICE a facility f which lies in a partition P prunes

at least as much area as pruned in six-region in P. Moreover, f can also prune space outside P as long as $maxAngle(f, P) < 90°$. Consider an example in Fig. 2.9(a), area pruned by f with SLICE is shaded while area in P_2 pruned with six-region is bounded by dotted arc. SLICE prunes larger space with higher number of partitions such as shown in Fig. 2.9(b), however increasing number of partitions will increase computational overhead. Therefore, choosing an appropriate number of partitions is important, which based on experiment the best performance is obtained with 12 partitions.



(a) SLICE prunes more area (b) Prunes larger space with more partitions

Figure 2.9: Comparison of SLICE and six-region [5]

In filtering, SLICE maintains k-smallest *upper arc* for each partition P which is called *bounding arc* (r_P^B). Any point in partition P and lies outside r_P^B fulfils $dist(p, q) > r_P^B$ which means that it is pruned by k facilities and therefore can be filtered from candidates of query answer. In verification of candidates, SLICE does not use boolean range query, instead it uses *significant facility* to verify whether a candidate point as RkNN of q. A facility point f is *significant facility* of partition P if its lower arc is smaller than bounding arc of P. In this situation, f can prune candidates in P. SLICE maintains an ascending ordered list of *significant facility* for each P called *sigList*. To verify a candidate user u, entries in the list is iteratively accessed. If a facility prunes u, a counter is incremented and if it equals to k, u is filtered and cannot be the query answer. The algorithm stops whenever it found that for a facility f currently accessed in *sigList*, $r_{f:P}^L > dist(u, q)$.

TPL++ [6] is the most current proposed method to solve RkNN query. It utilizes and improves TPL strategy with two optimization approaches. First, it improves filtering in TPL from $O(km)$ to $O(m)$ by proposing the use of disjoint and different size of subsets of facility to filter users. Consider an example in Fig. 2.10,

the entry e is filtered as it is pruned by two subsets $\{a, b\}$ and $\{c\}$.

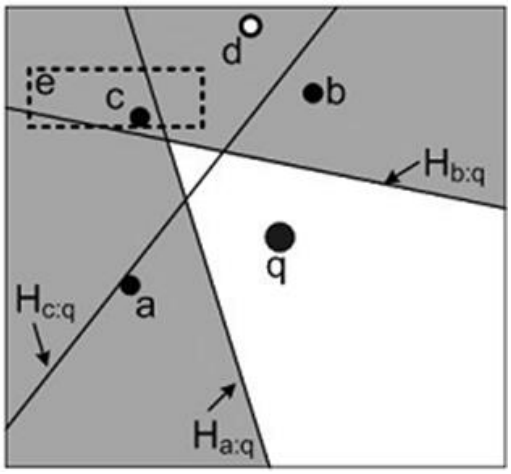


Figure 2.10: TPL++ optimized filtering [6]

The facilities in S_{fil} is iteratively accessed to prune an entry e. If e is pruned by accessed facilities, a counter is incremented. However, not all facilities can completely prune e, in this situation e^{tmp} is used to store part of e which cannot be pruned by a subset of facilities and will be initialized as e for next accessed subset. It continues until e^{tmp} becomes empty and the counter is incremented accordingly. If the counter equals to k than e can be filtered safely.

The second optimization is by including more facility points in S_{fil}. Consider Fig. 2.10, in TPL facility point d is not inserted in S_{fil} as it is pruned by other facilities, but TPL++ inserts it and does not perform check in point level. TPL++ only checks intermediate or leaf node whether it can be pruned or not. It increases filtering power and reduces IO and CPU cost by not checking facility point. This benefit outweigh the increasing filtering cost incurred by increasing size of S_{fil}.

A comprehensive experimental study of most notable RkNN was carried out in [6]. The comparison of computation complexity of each algorithm is shown in Table 2.2.1.

Table 2.1: Comparison of RkNN Algorithm Complexity [6]

Filtering Phase	**Six-reg**	**TPL**	**TPL++**	**FINCH**	**InfZone**	**SLICE**
Filter a f node	$O(1)$	$O(km)$	$O(m)$	$O(m)$	$O(m)$	$O(t)$
Filter a f point	$O(1)$	$O(km)$	$O(m)$	$O(logm)$	$O(m)$	$O(t)$
Add a f in S_{fil}	$O(logk)$	$O(logm)$	$O(logm)$	$O(m^2)$	$O(m^2)$	$O(tlogm)$
Verification Phase						
Filter a u node	$O(1)$	$O(km)$	$O(m)$	$O(m)$	$O(m)$	$O(t)$
Filter a u point	$O(1)$	$O(km)$	$O(m)$	$O(logm)$	$O(logm)$	$O(1)$
Expected candidates	$\frac{6k\|U\|}{\|F\|}$	$\frac{k\|U\|}{\|F\|}$ to $\frac{6k\|U\|}{\|F\|}$	$\frac{k\|U\|}{\|F\|}$ to $\frac{3.1k\|U\|}{\|F\|}$	$\frac{k\|U\|}{\|F\|}$ to $\frac{6k\|U\|}{\|F\|}$	$\frac{k\|U\|}{\|F\|}$	$< \frac{3.1k\|U\|}{\|F\|}$

f: facility, u: user, m: number of facilities in S_{fil}, t: number of partitions for SLICE, $|F|$: total number of facilities in the data set, $|U|$: total number of users in the data set

2.2.2 Reverse Top-k Query

Top-k query is a rank aware query which retrieves k objects considered by users as the best match of their preferences. While top-k query provides information for users that seek best products, reverse top-k query focuses on facilities' perspective who are interested to know the influenced users. Given a facility and a set of users with their scoring functions, a reverse top-k query returns users which consider the facility as one of their top k facilities [127].

Figure 2.11 illustrates users that are looking for the most important fuel stations based on their scoring functions. The scoring function covers two attributes, distance and price where lower value in these attributes is preferable. Top-2 fuel stations for all users are shown in the figure. The reverse top-2 of *Shell* are u_1, u_2 and u_3 because they consider *Shell* as one of their top-2 fuel stations.

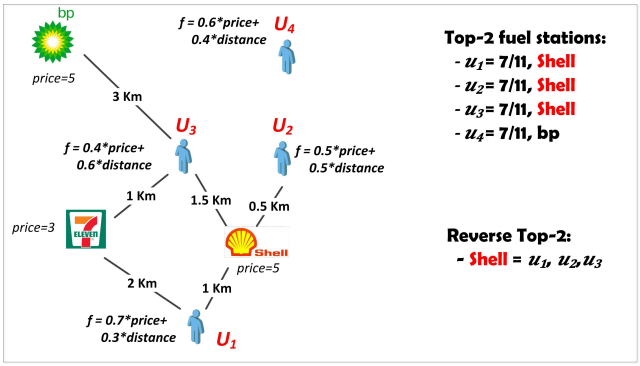


Figure 2.11: Illustration of Reverse Top-k

Reverse top-k Threshold Algorithm (RTA) [127] was the first algorithm to solve

reverse top-k query. Its main idea is to use threshold to avoid top-k query evaluation of all objects such happens in naive solution approach. It ignores preference weighting vectors which cannot contribute to the result. Reverse top-k computation was also used to identify the most m influential products on market [9]. Identifying the most influential objects is formulated as a query that retrieves m products with highest influence score. Reverse top-k computation was also utilized to monitor the popularity of location from the perspective of mobile users [128]. A study to solve multiple top-k queries [129] was also applicable for reverse top-k query. Branch and Bound (BBR) algorithm [130, 131] use score bounds on data object and MBR to provide high level pruning of weighting vectors.

Spatial reverse top-k queries consider the spatial distance between users and facilities. Spatial reverse top-k queries have been studied in Euclidean space, where distance is measured from the straight line that connecting two objects [132] and in road networks, where distance is computed as the total length of segments connecting two objects [133].

2.2.3 Reverse Skyline Query

A novel variant of skyline operator is *Reverse Skyline* Query (RSQ). An RSQ retrieves all users in a data set that have query point as a member of their skyline. RSQ was firstly introduced in [134] to observe the influence of a facility with respect to dominated facilities. The influence, in this case, intuitively can be defined as users for which the query facility is one of their skyline. In the example of Fig. 2.12, the reverse skyline of *7/11* are u_1, u_2 and u_3 because they all have *7/11* as a member of their skyline.

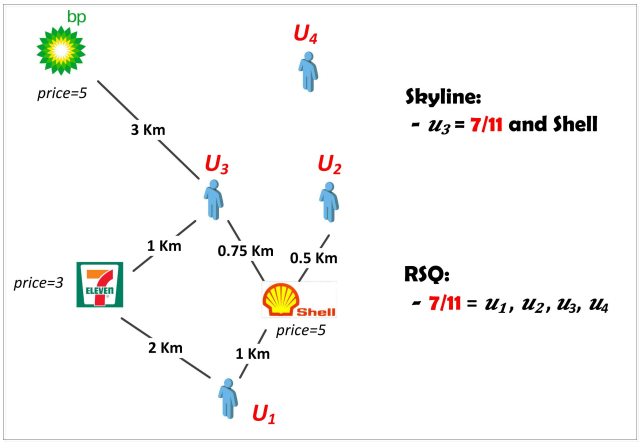


Figure 2.12: Illustration of Reverse Top-k

Many algorithms have been proposed to solve RSQ, including Branch and Bound Reverse Skyline, Reverse Skyline using Skyline Approximations [134], Full-Reuse-

Based Reverse Skyline and Global-Skyline-Based Reverse Skyline [135]. Some other algorithms were also proposed for solving RSQ for particular situations which are not relevant with our work. These algorithms including reverse skyline query for uncertain data [136], reverse skyline query with arbitrary non-metric similarity measures [137] and reverse skyline queries in wireless sensor networks [138]. Studies in Non traditional RSQ which are relevant with our topics include bichromatic reverse skyline [139] and that of answering why-not-questions in reverse skyline query [140] which defines safe region for moving reverse skyline query.

2.3 Continuous Monitoring of Spatial Queries

2.3.1 Continuous Range Query

Given a query q and a value of range r, a continuous range query is to continuously find all objects within distance r from q. Many algorithms have been proposed for continuously monitoring range queries. MobiEyes [24] delegates some computation load to the client side to reduce the server load during the monitoring of the queries. In [141], an adaptive indexing scheme is used to model the moving objects and queries. Kalashnikov *et al.* [142] uses grid structure to efficiently monitor continuous range queries.

Hu *et al.* [143] propose a safe zone for each object such that the results of the queries do not change if the object remains inside its safe zone. This algorithm is not applicable in the scenario when the queries are moving. A query indexing method called containment-encoded squares(CES)-based indexing is introduced in [144] to easily identify the moving objects that do not need to be evaluated. In [18], a safe zone is computed for each moving query to efficiently monitor the query results.

2.3.2 Continuous NN Query

Different from snapshot kNN, continuous kNN query continuously reports the real time k closest objects to the query when either query or objects continuously change their locations. Continuous kNN query has attracted significant research attentions in the past few years. Tao *et al.* [65] introduce time-parameterized queries (TP queries) with an assumption that motion pattern of the query is known. A TP query reports the NN of the query as well as its validity period. Zhang *et al.* [145] use TP queries to identify the safe zone of moving kNN queries. Some other works [146, 64, 147] employ grid structure to continuously monitor the kNN of queries.

2.3.3 Continuous Top-k Query

In Chapter 5, we study the continuous monitoring of moving top-k query. It continuously monitor the k most interesting facilities for a moving user with a particular scoring function.

The first work on moving top-k query was presented in [148]. The authors use the weighted distance as the ranking function and utilize multiplicatively weighted Voronoi cells to construct the safe zone for each query. They proposed Incremental Border Distance algorithm that prunes objects which cannot contribute to the safe zone. Huang *et al.* also proposed a safe zone based approach to monitor the result of moving top-k queries [149]. They use a more general ranking function compared to the ad-hoc one in [148]. These works were designed for the moving top$k$ spatial textual queries. They also consider the text relevancy of the objects, which is slightly above the scope of our study.

2.3.4 Continuous RkNN Query

The first algorithm for continuous monitoring of RNN queries was presented in . In this work, objects' velocities are assumed to be known. Xia *et al.* [116] proposed the first algorithm that does not assume any knowledge about the objects' motion pattern. This algorithm works based on six region approach and monitors the RNN queries by monitoring the unpruned area and the circles around the candidate objects. Consider the example of Fig. 2.13(a). The RNN of the query may change if one of the following cases occurs:

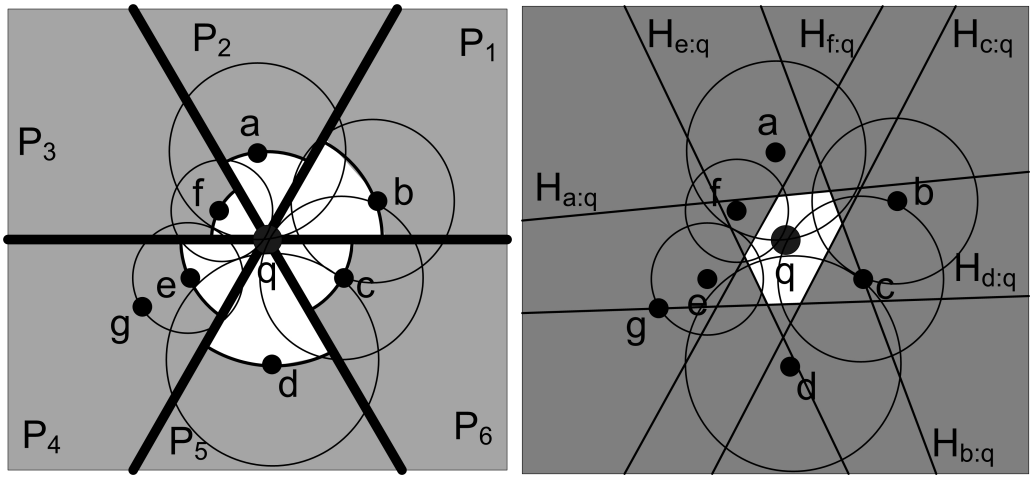
1. The query or one of the RNN candidates changes its location. In Fig. 2.13(a), this happens if either q, a, b, c, d, e or f moves to another location

2. The nearest neighbor of a candidate object is changed. In Fig. 2.13(a), it happens if a moving object enters or leaves the circles

3. A moving object enters the unpruned region (The region shown in white in Fig. 2.13(a)

Kang *et al.* [117] proposed RNN monitoring algorithm based on half-space (TPL) pruning approach that also monitors the unpruned area and the circles around the candidate objects. They use three observations above and apply grid structure to mark the monitored area. Consider the example of Fig. 2.13(b). The cells that overlap with the unpruned region and the monitored circles are marked. The movement of an object in a marked cell triggers the update of RNN.

The first RkNN monitoring algorithm was proposed by Wu *et al.* in [114]. In this algorithm, kNN queries are issued in each region and the users that are closer



(a) Six regions based monitoring [116] (b) Half-space based monitoring [117]

Figure 2.13: Continuous monitoring of RNN

than the k-th NN become the candidates and are verified if the query point is one of their k closest facilities. The RkNN of queries are monitored using the circle that contains k nearest facilities for each candidate object. Cheema *et al.* [118] proposed an algorithm, called Lazy Updates, that reduces the number of times pruning phase is executed. In this algorithm, they assign a safe region for each moving object such that if the object stays in the region, the pruning phase is not needed to be called.

Cheema *et al.* [5, 150] present the *state-of-the-art* algorithm for continuous RkNN queries. They introduce the notion of *influence zone* of a query q which is an area such that a user u is a RkNN of q if and only if u is inside this area. Thus, once the influence zone of a query is constructed, the system can efficiently monitor the RkNNs of a query by monitoring the users that enter or leave the query's influence zone.

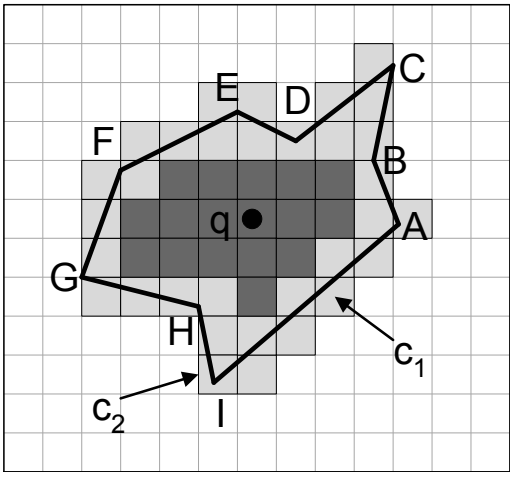


Figure 2.14: RkNN monitoring [5]

To efficiently monitor the users that enter or leave the influence zone, the space is partitioned using a grid containing $N \times N$ cells. A cell is called an *interior* cell if it is fully contained by the influence zone. A cell is called a *border* cell if it is partially contained by the influence zone. Consider polygon *ABCDEFGHI* in Fig. 2.14 which is the influence zone of q. The dark shaded cells are the *interior* cells and the light shaded ones are the *border* cells. For each cell, the algorithm maintains two lists called *interior list* and *border list*. The interior (resp. border) list of a cell c consists of every query q for which the cell c is an interior cell (resp. a border cell). Whenever a user u enters a cell c, it becomes the RkNNs of every query in the interior list of c. The algorithm also checks every query in the border list of c and checks whether u is inside its influence zone or not and updates the results accordingly.

2.3.5 Reducing Communication Cost

Continuous monitoring of spatial queries have been studied in various types of query, such as in nearest neighbour (NN) query [113, 145, 70, 151, 152, 146, 147, 153], range query [154, 25, 24, 26, 27, 18] and reverse nearest neighbour (RNN) query [66, 68, 116, 117, 114, 5, 155]. Papadias *et al.* [146] presented conceptual partitioning (CPM) to continuously monitor the nearest neighbour of queries. It uses a circle C to maintain the query result. The result is re-computed from the scratch if the NNs that move outside C are more than the outer objects that move into C. Otherwise, the k objects inside C that are closest to the query are set as the result.

Most of the existing works for continuous monitoring of spatial queries focus on reducing computation cost, and reducing communication cost does not receive considerable attentions. Some of the works that focus on reducing communication cost include [24], [143], [156], [118] and [157]. Gedik *et al.* [24] proposed MobiEyes to reduce the computation and communication cost of range queries monitoring. MobiEyes delegates some computation load to the client objects. Wang *et al.* [26] introduced the concept of query view such that the query results are affected only if there is object that changes its current query view. A distributed server infrastructure was presented in [27] to partition the service region into several zones that can cooperatively monitor the results of reverse nearest neighbour queries. Hu *et al.* [143] proposed a generic framework to continuously monitor the result of range and kNN queries over moving object. For each object, they define a safe zone such that the result of all queries remain unchanged as long as the object lies inside the safe zone. The framework was designed to handle range and nearest neighbour queries.

In [156], a threshold-based algorithm was presented to handle the continuous monitoring of nearest neighbour of queries with minimum communication overhead. Cheema *et al.* [118] and Emrich *et al.* [157] introduced a rectangular shaped of safe

zone for each moving object to minimize the communication cost between clients and server in the continuous reverse nearest neighbour queries. Note that these studies were designed to handle specific type of spatial queries. In Chapter 6, we propose a generic framework that can be easily applied to handle many types of spatial queries.

Chapter 3

Reverse Approximate Nearest Neighbor Queries (RANN)

In this chapter, we introduce an alternative definition of influence by considering relative distance between users and facilities. It complements the influence definition in reverse k nearest neighbor queries (RkNN) that uses relative ordering of facilities based on their distances from the users.

3.1 Introduction

People usually prefer the facilities in their vicinity, i.e., they are influenced by nearby facilities. A *reverse nearest neighbors* (RNN) query [10, 112, 119, 158] aims at finding every user that is influenced by a query facility q. Formally, given a set of users U, a set of facilities F and a query facility q, an RNN query returns every user $u \in U$ for which the query facility q is its closest facility. The set containing RNNs, denoted as $RNN(q)$, is also called the influence set of q.

Consider the example of Fig. 3.1 that shows four Starbucks cafes (f_1 to f_4) and three users (u_1 to u_3). In the context of RNN queries, the users u_2 and u_3 are both influenced by f_1 because f_1 is their closest Starbucks. Therefore, u_2 and u_3 are the RNNs of f_1, i.e., $RNN(f_1) = \{u_2, u_3\}$. Similarly, it can be confirmed that $RNN(f_2) = \emptyset$, $RNN(f_3) = \emptyset$, $RNN(f_4) = \{u_1\}$.

A *reverse k nearest neighbors* (RkNN) query [4, 3, 159, 5, 155, 6] is a natural extension of the RNN query. Specifically, in the context of an RkNN query, a user u is considered to be influenced by its k closest facilities. Hence, an RkNN query q returns every user $u \in U$ for which q is among its k closest facilities. In the example of Fig. 3.1, assuming $k = 2$, $R2NN(f_2) = \{u_1, u_2, u_3\}$ because f_2 is one of the two closest facilities for all of the three users. Similarly, $R2NN(f_1) = \{u_2, u_3\}$, $R2NN(f_3) = \emptyset$ and $R2NN(f_4) = \{u_1\}$.

RkNN queries have numerous applications [10] in location based services, resource allocation, profile-based management, decision support etc. Consider the example of a supermarket. The people for which this supermarket is one of the k closest supermarkets are its potential customers and may be influenced by targeted marketing or special deals. Due to its significance, RNN queries and its variants have received significant research attention in the past decade (see [3] for a survey).

In this chapter, we propose an alternative definition of influence and propose a variant of RNN queries called *reverse approximate nearest neighbors* (RANN) query. This definition is motivated by our observation that an RkNN query may not properly capture the notion of influence.

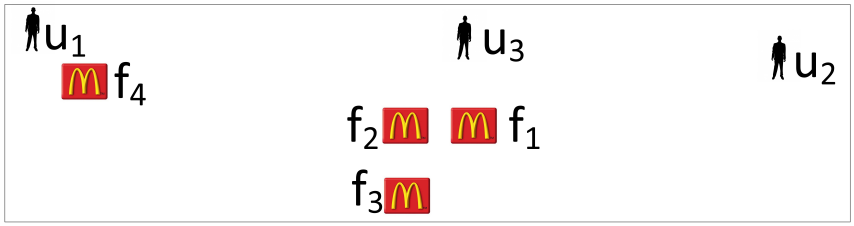


Figure 3.1: Illustration of RNN query and its variants

3.1.1 Motivation

Consider the example of a person living in a suburban area (e.g., u_2 in Fig. 3.1) who does not have any Starbucks nearby. Her nearest Starbucks is f_1 which is say 30 Km from her location. In the context of R2NN query, u_2 is influenced by f_1 and f_2 – her two nearest facilities. However, we argue that it is also influenced by f_3 because a user who needs to travel a minimum of 30 Km to visit a Starbucks may also be willing to travel to a Starbucks cafe 31 Km far from her.

Similarly, consider the example of another person living in a suburb (e.g., u_1 in Fig. 3.1) who has only one Starbucks nearby (f_4) assuming that all other Starbucks (e.g., f_1 to f_3) are in downtown area and are quite far. In the context of R2NN queries, the user u_1 is considered to be influenced by both f_4 and f_2 because these are her two closest facilities. However, we argue that the user u_1 is only influenced by f_4 because the other facilities are significantly farther than $dist(u_1, f_4)$, e.g., a user who has a Starbucks within 1 Km is not very likely to visit a Starbucks that is say 30 Km from her location.

As shown above, the definition of influence used in RkNN queries considers only the relative ordering of the facilities based on their distances from u and ignores the actual distances of the facilities from u. Motivated by this, in this chapter, we propose a reverse approximate nearest neighbors (RANN) query that relaxes the definition of influence using a parameter x (called the x factor) and considers the relative distances between users and facilities. Specifically, an RANN query returns

every user u for which the query facility is its approximate nearest neighbor.

Definition 3.1 ***Approximate nearest neighbor***. *Let $NNdist(u)$ denote the distance between u and its nearest facility. Given a value of $x > 1$, a facility f is called an approximate nearest neighbor of u if $dist(u, f) \leq x \times NNdist(u)$.*

Reverse Approximate Nearest Neighbors (RANN) query. Given a value of $x > 1$, an RANN query q returns every user u for which $dist(u, q) \leq x \times NNdist(u)$, i.e., return every user u for which q is its approximate nearest neighbor. The set of RANNs of a query q is denoted as $RANN_x(q)$. Note that an RANN query is the same as an RNN query if $x = 1$.

In the example of Fig. 3.1, assuming $x = 1.2$, RANN of f_2 are the users u_2 and u_3, i.e., $RANN_{1.2}(f_2) = \{u_2, u_3\}$. Similarly, $RANN_{1.2}(f_1) = \{u_2, u_3\}$, $RANN_{1.2}(f_3) = \{u_2\}$ and $RANN_{1.2}(f_4) = \{u_1\}$.

Remark. RkNN queries and RANN queries assume that the distance is the main factor influencing a user. This assumption holds in many real world scenarios. For instance, the users looking for nearby fuel stations are usually not concerned about price (or even rating) because all fuel stations have similar price (or even the same price because, in some countries, the fuel prices are regulated by the government). Similarly, users interested in McDonald's restaurants or Starbucks cafe are mainly influenced by the distance because other attributes such as price, menu, and ratings are the same for all stores. Nevertheless, in the case where the users are influenced by other attributes, reverse top-k queries [127, 160, 132] can be used to compute the influence using a scoring function involving multiple attributes such as distance, price, and rating. This is a different line of research and is not within the scope of this chapter.

3.1.2 Contributions

We make the following contributions in this chapter.

1. We complement the RkNN queries by proposing a new definition of influence that considers every user u to be influenced by a query q for which q is an approximate nearest neighbor.

2. As we show in Section 3.3, the pruning techniques used to solve RkNN queries cannot be applied or extended for RANN queries. This is mainly because, in our problem settings, a facility f may not be able to prune the users that are quite far from f (see Section 3.3 for details). Based on several non-trivial observations, we propose efficient pruning techniques that are proven to be *tight*, i.e., given a facility f used for pruning, the pruning techniques guarantee to prune every point that can be pruned by f. We then propose an efficient algorithm that utilizes these pruning techniques to efficiently answer the snapshot RANN queries.

3. We conduct an extensive experimental study on three real data sets and several synthetic data sets to show the effectiveness of our proposed techniques. Since existing techniques cannot be extended to answer RANN queries, we compare our algorithm with a naïve algorithm (called RQ) as well as a significantly improved version of RQ (called IRQ). The experimental results show that our algorithm is several orders of magnitude better than both of the competitors. Furthermore, we note that the results of an RANN query are the same as the RkNN ($k = 1$) query when x is quite close to 1. Therefore, we also compare our algorithm (by setting $x = 1 + 10^{-0.6}$) with the most notable RNN algorithms. Although our algorithm solves a more challenging version of the problem, our experiments show that it performs reasonably well compared to RNN algorithms.

3.2 Problem Definition

Similar to RkNN queries, RANN queries can also be classified into *bichromatic* RANN queries and *monochromatic* RANN queries.

Bichromatic RANN query. Given a set of users U, a set of facilities F, a query facility q (which may or may not be in F), and a value of $x > 1$, a bichromatic RANN query returns every user $u \in U$ for which $dist(u, q) \leq x \times NNdist(u)$ where $NNDist(u)$ denotes the distance between u and its nearest facility in F.

Monochromatic RANN query. Given a set of facilities F, a query facility q (which may or may not be in F), and a value of $x > 1$, a monochromatic RANN query returns every facility $f \in F$ for which $dist(f, q) \leq x \times NNdist(f)$ where $NNDist(f)$ denotes the distance between f and its nearest facility in $\{F - f\}$.

In Fig. 3.1, the monochromatic RANNs of f_2 (assuming $x = 1.5$) are f_1 and f_3. Monochromatic queries aim at finding the facilities that are influenced by the query facility. Consider a set of police stations. For a given police station q, a monochromatic query returns the police stations for which q is a nearby police station. Such police stations may seek assistance (e.g., extra policemen) from q in case of an emergency event.

Although our techniques can be easily applied to monochromatic RANN queries, in this chapter, we focus on bichromatic RANN queries because the bichromatic version has more applications in real world scenarios. Similar to the existing work on RNN queries, we assume that both the facility and user data sets are indexed by R*-tree [161]. The R*-tree that indexes the set of facilities (resp. users) is called facility (resp. user) R*-tree. Since most of the applications of the RNN query and its variants are in location-based services, similar to the existing RNN algorithms [3], the focus of this chapter is on two dimensional location data.

3.3 Pruning Techniques

Given a facility f, a user u cannot be the RANN of q if $dist(u, q) > x \times dist(u, f)$. In such case, we say that the facility f prunes the user u. In this section, we will present the pruning techniques that use a facility f or an MBR of the facility R*-tree to prune the users. First, we highlight the challenges.

3.3.1 Challenges

Existing pruning techniques cannot be applied or extended for the RANN queries due to the unique challenges involved. For instance, the algorithms to solve RkNN queries can prune most of the search space by considering only the nearby facilities surrounding q. Consider the example of Fig. 3.3 where the six-regions approach finds the nearest facility to the query q in each of the six partitions and the shaded area can be pruned.

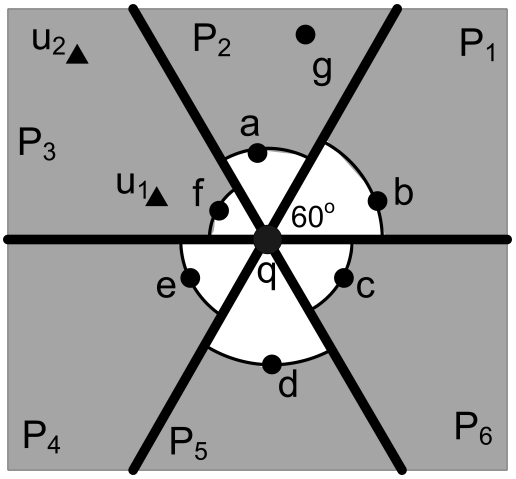


Figure 3.2: Six-regions pruning

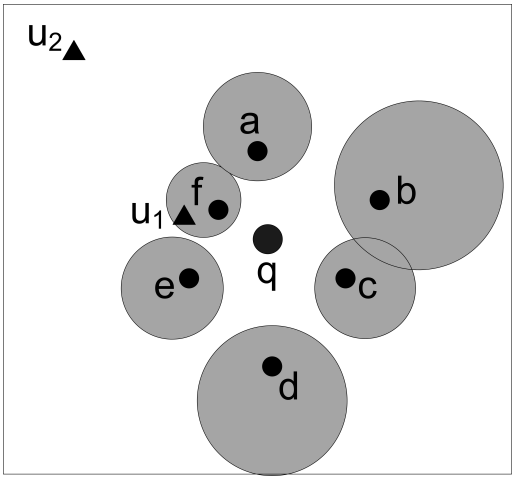


Figure 3.3: RANN pruning challenges

However, in the case of RANN queries, the nearby facilities surrounding the query q are not sufficient to prune a large part of the search space. Assuming $x = 2$, in partition P_3 (see Fig. 3.3), while the user u_1 can be pruned by f the user u_2 cannot be pruned by f. In other words, the users that are further from a facility f are less likely to be pruned by it.

In Fig. 3.3, assuming $x = 2$, the six shaded circles show the maximum possible area that can be pruned by the six facilities a to f (the details on how to compute the circles will be presented later). Note that the facilities that are close to q prune a smaller area as compared to the farther facilities. Hence, the algorithm needs to access not only nearby facilities but also farther facilities to prune a large part of the search space. Also, note that RANN queries are more challenging because the

maximum area that can be pruned is significantly smaller.

In Section 3.3.2, we present the pruning techniques that prune the space using a data point, i.e., a facility f. In Section 3.3.3, we present the techniques to prune the space using an MBR of the facility R*-tree. Efficient implementation of the pruning techniques is discussed in Section 3.3.4.

3.3.2 Pruning using a facility point

Before we present our non-trivial pruning technique, we present the definition of a pruning circle.

Definition 3.2 (Pruning circle) *Given a query q, a multiplication factor $x > 1$ and a point p, the pruning circle of p (denoted as C_p) is a circle centered at c with radius r where $r = \frac{x \cdot dist(q,p)}{x^2-1}$ and c is on the line passing through q and p such that $dist(q,c) > dist(p,c)$ and $dist(q,c) = \frac{x^2 \cdot dist(q,p)}{x^2-1}$.*

Consider the example of Fig. 3.4 that shows the pruning circle C_f of a facility f assuming $x = 2$. The centre of c is located on the line passing through q and f such that $dist(q,c) = \frac{4 \cdot dist(q,f)}{3}$, $dist(q,c) > dist(f,c)$ and radius $r = \frac{2 \cdot dist(q,f)}{3}$. The condition $dist(q,c) > dist(f,c)$ ensures that c lies towards f on the line passing through q and f, i.e., f lies between the points c and q as shown in Fig. 3.4. Next, we introduce our first pruning rule in Lemma 3.1.

Lemma 3.1 *Every user u that lies in the pruning circle C_f of a facility f cannot be the RANN of q, i.e., $dist(u,q) > x \times dist(u,f)$.*

Proof Given two points v and w, we use $\overline{vw}$ to denote $dist(v,w)$. Consider the example of Fig. 3.4. Since u is inside the circle C_f, $\overline{uc} < r$. Assume that $\overline{uc} = n \cdot r$ where $0 \leq n < 1$. Since $r = \frac{x \cdot \overline{qf}}{x^2-1}$, we have $\overline{uc} = n \cdot r = n \cdot \frac{x \cdot \overline{qf}}{x^2-1}$.

Considering the triangle $\triangle quc$, $\overline{qu} = \sqrt{(\overline{qc})^2 + (\overline{uc})^2 - 2 \cdot \overline{uc} \cdot \overline{qc} \cdot \cos\theta}$. Since $\overline{uc} = n \cdot \frac{x \cdot \overline{qf}}{x^2-1}$ and $\overline{qc} = \frac{x^2 \cdot \overline{qf}}{x^2-1}$, we have

$$\begin{aligned} \overline{qu} &= \sqrt{(\frac{x^2 \cdot \overline{qf}}{x^2-1})^2 + n^2(\frac{x \cdot \overline{qf}}{x^2-1})^2 - 2n(\frac{x \cdot \overline{qf}}{x^2-1})(\frac{x^2 \cdot \overline{qf}}{x^2-1}) \cdot \cos\theta} \\ &= \sqrt{(\frac{x \cdot \overline{qf}}{x^2-1})^2(x^2 + n^2 - 2 \cdot x \cdot n \cdot \cos\theta)} \\ &= (\frac{x \cdot \overline{qf}}{x^2-1})\sqrt{x^2 + n^2 - 2xn\cos\theta} \end{aligned} \tag{3.1}$$

Similarly considering $\triangle fcu$, $\overline{fu} = \sqrt{(\overline{fc})^2 + (\overline{uc})^2 - 2 \cdot \overline{uc} \cdot \overline{fc} \cdot \cos\theta}$. Since $\overline{fc} = \overline{qc} - \overline{qf}$ and $qc = \frac{x^2 \cdot \overline{qf}}{x^2-1}$, we get $\overline{fc} = \frac{\overline{qf}}{x^2-1}$. We can obtain the value of $\overline{fu}$ by replacing the values of $\overline{fc}$ and $\overline{uc}$.

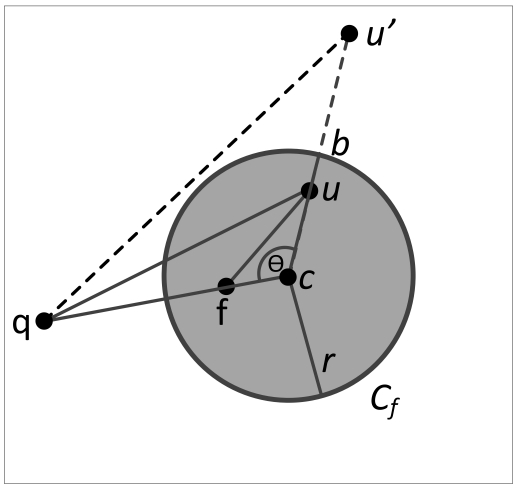


Figure 3.4: Lemma 3.1

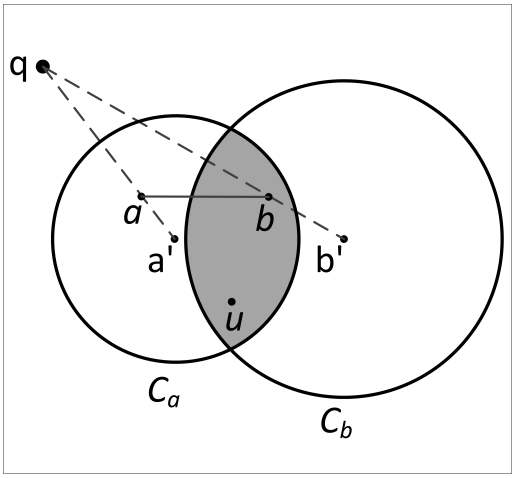


Figure 3.5: Lemma 3.3

$$\begin{aligned}\overline{fu} &= \sqrt{(\frac{\overline{qf}}{x^2-1})^2 + n^2(\frac{x\cdot\overline{qf}}{x^2-1})^2 - 2\cdot n(\frac{x\cdot\overline{qf}}{x^2-1})\cdot(\frac{\overline{qf}}{x^2-1})\cdot\cos\theta} \\ &= (\frac{\overline{qf}}{x^2-1})\sqrt{1+n^2x^2-2nx\cos\theta}\end{aligned} \tag{3.2}$$

Note that the user u can be pruned if $dist(u,q) > x \times dist(u,f)$. Therefore, we need to show $\overline{qu} - x\cdot\overline{fu} > 0$. The left side of this inequality can be obtained using the values of $\overline{qu}$ and $\overline{fu}$ from Eq. (3.1) and Eq. (3.2), respectively.

$$\overline{qu} - x\cdot\overline{fu} = \frac{x.\overline{qf}}{x^2-1}(\sqrt{x^2+n^2-2xn\cos\theta} - \sqrt{1+x^2n^2-2xn\cos\theta}\ ) \tag{3.3}$$

Since $x > 1$, $(\frac{x.\overline{qf}}{x^2-1})$ is always positive. Hence, we just need to prove that $(\sqrt{x^2+n^2-2xn\cos\theta} - \sqrt{1+x^2n^2-2xn\cos\theta} > 0$. In other words, we need to show $(\sqrt{x^2+n^2-2xn\cos\theta} > \sqrt{1+x^2n^2-2xn\cos\theta}$. Note that both sides of this inequality are positive (otherwise $\overline{qu}$ and $\overline{fu}$ in Eq. (3.1) and Eq. (3.2) would be negative which is not possible). Hence, we can take the square of both sides resulting in $x^2+n^2-2xn\cos\theta > 1+x^2n^2-2xn\cos\theta$ which implies that we need to prove $(x^2+n^2-x^2n^2-1) > 0$. This inequality can be simplified as $(x^2-1)(1-n^2) > 0$. Since $x > 1$ and $n < 1$, it is easy to see that $(x^2-1)(1-n^2) > 0$ which completes the proof. ∎

Note that although the pruning technique itself is non-trivial, applying this pruning rule is not expensive, i.e., to check whether a user u can be pruned or not, we only need to compute its distance from the centre c and compare it with the

radius r. Next, we show that this pruning rule is *tight* in the sense that any user u' that lies outside C_f is guaranteed not to be pruned by the facility f.

Lemma 3.2 *Given a facility f and a user u' that lies on or outside its pruning circle C_f, then $dist(u', q) \leq x \times dist(u', f)$, i.e., u' cannot be pruned by f.*

Proof Consider the user u' in Fig. 3.4. Since u' is on or outside the pruning circle, it satisfies $\overline{u'c} = n \cdot r$, where $n \geq 1$. The proof is similar to the proof of Lemma 3.1 except that we need to show that $\overline{u'q} - x.\overline{fu'} \leq 0$, i.e., we need to show $(x^2 - 1)(1 - n^2) \leq 0$ which is obvious given that $x > 1$ and $n \geq 1$. ∎

Note that the pruning circle C_f is larger if $dist(q, f)$ is larger which implies that the facilities that are farther from the query prune larger area. For instance, in Fig. 3.5, the pruning circle C_b is bigger than the pruning circle C_a.

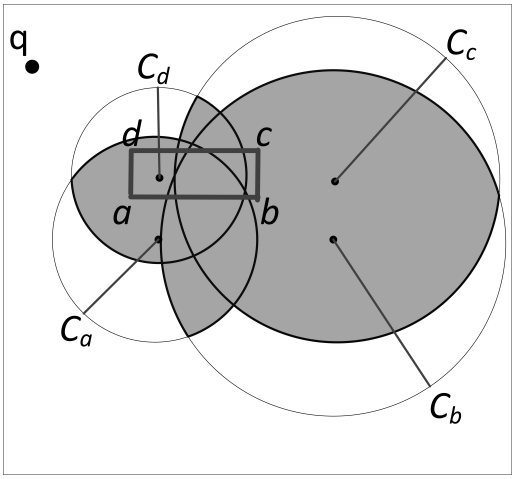


Figure 3.6: Pruning using MBR

3.3.3 Pruning using the nodes of facility R*-tree

In this section, we present our techniques to prune the search space using the intermediate or leaf nodes of the facility R*-tree. These pruning techniques reduce the I/O cost of the algorithm because the algorithm may prune the search space using a node of the R*-tree instead of accessing the facilities in its sub-tree.

A node of the facility R*-tree is represented by a minimum bounding rectangle (MBR) that encloses all the facilities in its sub-tree. Without accessing the contents of the node, we cannot know the locations of the facilities inside it except that each side of the MBR contains at least one facility. We utilize this information to devise our pruning techniques. Specifically, we use all four sides of the MBR and use each side (i.e., line segment) to prune the search space. Lemma 3.3 presents the pruning rule and Fig. 3.5 provides an illustration.

Lemma 3.3 *Given a query q, a multiplication factor $x > 1$, and a line $\overline{ab}$ representing a side of an MBR, a user u cannot be the RANN of q if it lies inside both of the pruning circles C_a and C_b, i.e., u can be pruned if u lies in $C_a \cap C_b$.*

Proof Let $maxdist(p, \overline{ab})$ denote the maximum distance between a point p and a line $\overline{ab}$. Note that $maxdist(u, \overline{ab}) = max(dist(u, a), dist(u, b))$. Since u lies in both C_a and C_b, $dist(u, q) > x \times dist(u, a)$ and $dist(u, q) > x \times dist(u, b)$ (according to Lemma 3.1). In other words, $dist(u, q) > x \times maxdist(u, \overline{ab})$. Since there is at least one facility f on the line $\overline{ab}$, $dist(u, f) \leq maxdist(u, \overline{ab})$. Hence, $dist(u, q) > x \times dist(u, f)$ which implies that the user u can be pruned. ∎

In Fig. 3.5, the shaded area can be pruned by using the line $\overline{ab}$. The next lemma shows that this pruning rule is also tight.

Lemma 3.4 *Given a line $\overline{ab}$ such that the only information we have is that there is at least one facility f on $\overline{ab}$, a user u cannot be pruned if it lies outside either C_a or C_b.*

Proof Without the loss of generality, assume that u lies outside C_a. Now assume that there is exactly one facility f on the line $\overline{ab}$ and it lies at the end point a. Since f lies on a, $C_a = C_f$ which implies that u is outside C_f. Hence, u cannot be pruned by f (Lemma 3.2). ∎

To prune the search space using an MBR, we apply Lemma 3.3 on each of side s_i of the MBR. Specifically, a user u can be pruned if, for *any* side s_i of the MBR, u lies in *both* of the pruning circles of the end points of s_i. Consider the example of Fig. 3.6 where an MBR *abcd* is shown along with the pruning circles of the corners of the MBR (see C_a to C_d). Let A_i denote the area pruned by a side s_i of the MBR. In Fig. 3.6, the shaded area can be pruned which corresponds to $\cup_{i=1}^{4} A_i$ where $A_1 = C_a \cap C_b$, $A_2 = C_b \cap C_c$, $A_3 = C_c \cap C_d$, and $A_4 = C_d \cap C_a$.

3.3.4 Implementation of the pruning techniques

In the previous sections, we discussed how to prune the search space using a facility point or an MBR of the facility R*-tree. In this section, we discuss how to efficiently and effectively implement the pruning techniques.

Assume that we have a set of facilities and MBRs to be used for pruning the search space. Let A_i denote the area pruned by a facility point or a side of an MBR. Let $\mathcal{A} = \{A_i, \cdots, A_n\}$ be the total area that can be pruned by using the set of facilities and MBRs. In this section, we present Algorithm 1 that efficiently checks whether an entry e of user R*-tree (i.e., a point or an MBR) can be pruned by $\mathcal{A}$ or

not, i.e., whether e lies inside $\mathcal{A}$ or not. Before we discuss the details of Algorithm 1, we describe how to prune a user MBR e using a single pruning area $A_i \in \mathcal{A}$. Since e is an MBR, it is possible that e only partially lies in A_i. Ideally, we should be able to prune the part of the MBR that lies inside A_i. In our algorithm, we process the MBR e such that the area that lies inside A_i is trimmed. Below are the details on how to do this.

Case 1: A_i corresponds to the area pruned by a facility. Consider the example of Fig. 3.7 where A_i corresponds to the circle C_a. Note that only a part of the rectangle R lies in the circle. In such case, we conservatively approximate the area that can be pruned. Specifically, we use a function `TrimEntry`(C_a, R) that trims the MBR R using a circle C_a and returns R_a that corresponds to the minimum bounding rectangle of the part of R that lies outside C_a, i.e., R_a cannot be pruned by C_a. In Fig. 3.7, R_a is the shaded area. In Fig. 3.8, R_b (the light shaded area) is returned by `TrimEntry`(C_b, R). The function `TrimEntry`(C_a, R) can be implemented as follows. Let I be the set of intersection points between a circle C_a and a rectangle R. Let $\mathcal{C}$ be the corners of R that lie outside C_a. The trimmed entry R_a is the minimum bounding rectangle enclosing the points in $I \cup \mathcal{C}$.

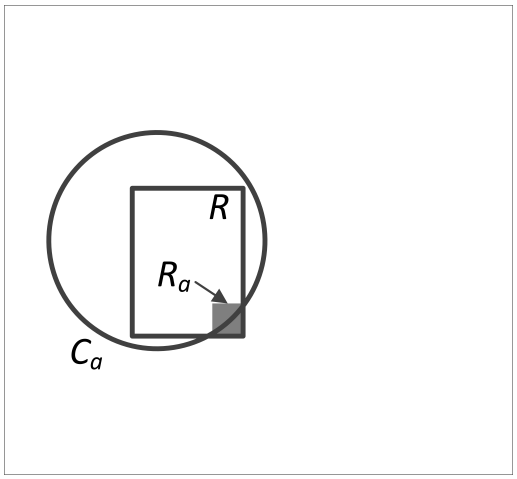


Figure 3.7: Trimming an MBR

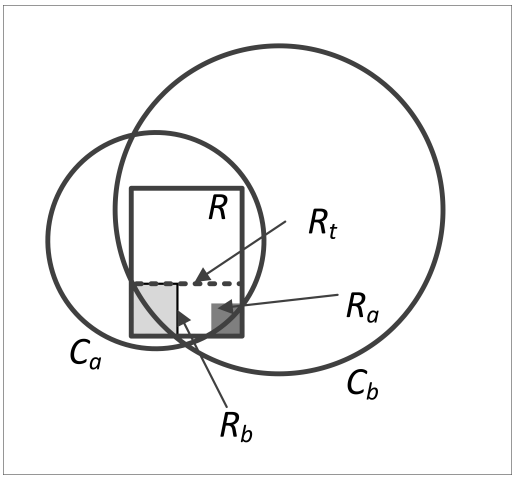


Figure 3.8: Pruning an entry

Case 2: A_i corresponds to the area pruned by a side of an MBR. Consider the example of Fig. 3.8 where A_i corresponds to the area pruned by a line $\overline{ab}$, i.e., $A_i = C_a \cap C_b$. In this case, we find the part of the MBR R that cannot be pruned by A_i as follows. Let R_a =TrimEntry(C_a, R) (see the dark shaded area) and R_b=TrimEntry(C_b, R) (see the light dotted area) in Fig. 3.8. The unpruned part of R is the minimum bounding rectangle enclosing both R_a and R_b, e.g., R_t shown in thick broken lines in Fig. 3.8 cannot be pruned by $C_a \cap C_b$.

Algorithm 1 shows the details of how to prune an entry e using a set of pruned areas $\mathcal{A}$. The output of the algorithm is the part of e that cannot be pruned by $\mathcal{A}$.

Algorithm 1 PruneEntry($e, \mathcal{A}$)

Input: e: the entry to be pruned, $\mathcal{A}$: the set of pruned areas
Output: Return the part of e that cannot be pruned by $\mathcal{A}$

```
1: for each A_i ∈ A do
2:   if A_i is related to a facility f then
3:     R ← TrimEntry(C_f,e)
4:   else if A_i is related to a line ab then
5:     R_a ← TrimEntry(C_a,e)
6:     R_b ← TrimEntry(C_b,e)
7:     R ← minimum bounding rectangle enclosing both R_a and R_b
8:   e ← R
9:   if e is empty then
10:    return φ
11: return e
```

Each entry A_i is iteratively accessed from $\mathcal{A}$ and the entry e is trimmed using the details described earlier (lines 2 to line 7). The trimmed part R is assigned to e which is to be further trimmed in the next iteration (line 8). At any stage, if e is empty, the algorithm terminates by returning ϕ (line 10) which indicates that the whole entry e can be pruned by $\mathcal{A}$. When all entries A_i in $\mathcal{A}$ have been accessed, the algorithm returns e.

We remark that although the trimming significantly improves the I/O cost (2 to 3 times) of the algorithm, the overall CPU time is also increased due to the overhead of trimming. This must be taken into consideration when making the decision on whether to use trimming or not, e.g., the trimming should not be used if the main focus is to optimize CPU cost.

Improving Algorithm 1. Note that Algorithm 1 accesses every entry $A_i \in \mathcal{A}$ regardless of whether A_i can prune a part of e or not. Now, we discuss how to improve the efficiency of Algorithm 1 by ignoring the entries A_i that cannot prune e. Similar to six-regions approach [112] and SLICE [6], we divide the whole space around q in t equally sized partitions, e.g., see the partitions P_1 to P_6 in Fig. 3.9. Our technique is based on the following two simple observations.

Observation 1. Let $\mathcal{P}$ be the set of partitions overlapped by a pruned area A_i. An entry e can be pruned by A_i only if e overlaps with at least one partition in $\mathcal{P}$. Consider the example of Fig. 3.9 where the area A_i is shown shaded and overlaps with partitions P_3 and P_4. Since the entry e_1 does not overlap with P_3 or P_4, it cannot be pruned by A_i.

Observation 2. Let $A_i.max$ and $A_i.min$ denote the maximum and minimum distances between q and the pruned area A_i, respectively. Fig. 3.9 shows $A_i.max = dist(q, a)$ and $A_i.min = dist(q, b)$. We remark that $A_i.max$ and $A_i.min$ can be computed following the ideas presented in [162, 18]. Note that an entry e cannot be pruned by

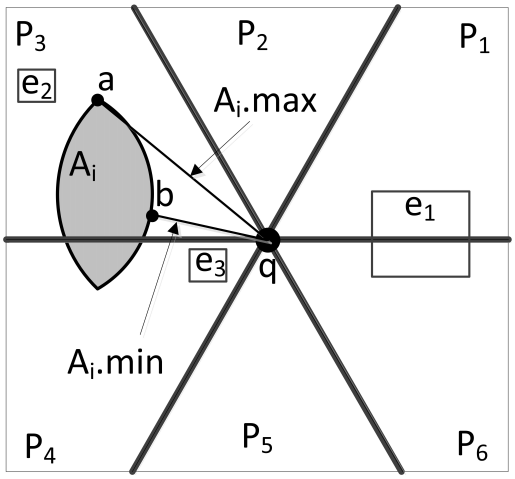


Figure 3.9: Observations 1&2

A_i if $mindist(q, e) > A_i.max$ or $maxdist(q, e) < A_i.min$. For instance, the entry e_2 cannot be pruned by A_i because $mindist(q, e_2) > A_i.max$. Similarly, the entry e_3 cannot be pruned because $maxdist(q, e_3) < A_i.min$.

Let $A_i.interval$ denote an interval from $A_i.min$ to $A_i.max$ and $e.interval$ denote an interval from $mindist(q, e)$ to $maxdist(q, e)$. Observation 2 shows that an entry e can be pruned by A_i only if $e.interval$ overlaps with $A_i.interval$. We use an interval tree [163] to efficiently retrieve every A_i for which $A_i.interval$ overlaps with $e.interval$. Specifically, for each partition P_i, we maintain an interval tree $\mathcal{T}_i$ that contains $A_j.interval$ for every $A_j \in \mathcal{A}$ that overlaps with P_i. To check whether an entry e (that overlaps with a partition P_i) can be pruned by $\mathcal{A}$, we issue an interval query on $\mathcal{T}_i$ with input interval $e.interval$. Let $\mathcal{A}_e$ denote the set containing every area A_j returned by the interval query $e.interval$. In Algorithm 1, we use $\mathcal{A}_e$ instead of $\mathcal{A}$. Note that the cost of interval query is $O(m + \log n)$ where n is the number of intervals stored in the interval tree and m is the number of intervals that overlap with the input interval.

3.4 Algorithm

Our algorithm consists of three phases namely *pruning*, *filtering* and *verification*. In the pruning phase, we use the facility R*-tree to prune the search space, i.e., compute $\mathcal{A}$. In the filtering phase, the users that lie in the pruned space are pruned and the remaining users are inserted in a candidate list called L_{cnd}. Finally, in the verification phase, each candidate user $u \in L_{cnd}$ is verified to check whether it is a RANN of q or not.

Pruning Phase Algorithm 2 presents the details of the pruning phase. The algorithm initializes a heap h with the root of the facility R*-tree. The entries are

iteratively de-heaped from the heap and are processed as follows. If a de-heaped entry e is pruned (i.e., the entry e' returned by Algorithm 1 is empty), we ignore it (lines 5 and 6). Otherwise, we process it as follows.

Algorithm 2 Pruning

Input: facility R*-tree, and a query q
Output: The set of pruned areas $\mathcal{A}$

```
1:  A ← φ
2:  insert root of facility R-tree in a h
3:  while h is not empty do
4:    de-heap an entry e
5:    e' ← PruneEntry(e, A)                          ▷ Algorithm 1
6:    if e' ≠ φ then                                 ▷ e is not pruned
7:      if e is an intermediate or leaf node then
8:        for each side ab of e do
9:          create A_i = C_a ∩ C_b and insert in A
10:       for each child c of e do
11:         if c overlaps with e' then insert c in the heap
12:     else                                         ▷ e is a facility point
13:       create A_i = C_e and insert in A
```

If e is an intermediate or leaf node of the R*-tree, for each side of e, we create a pruning area A_i and insert it in $\mathcal{A}$ (line 9). We also insert its children in the heap h. Note that a child c of e that does not overlap with e' can be pruned because it lies in the pruned area. Hence, only the children that overlap with e' are inserted in the heap (line 11). If e is a facility point, we create the pruning circle C_e and add it to $\mathcal{A}$ (line 13). The algorithm terminates when the heap becomes empty.

Filtering Phase Algorithm 3 describes the filtering phase. A stack S is initialized with the root entry of the user R*-tree. Each entry e is iteratively retrieved from S and processed as follows. If e can be pruned by $\mathcal{A}$, it is ignored (lines 5 and 6). Otherwise, if it is an intermediate or leaf node, its children that overlap with e' are inserted in the stack (line 9). If e is a user, it is inserted in L_{cnd} (line 11). The algorithm stops when the stack S becomes empty.

Verification Phase In the verification phase, each candidate user $u \in L_{cnd}$ is verified as follows. Note that a user u is a RANN if and only if there is no facility f for which $dist(u, f) < \frac{dist(u,q)}{x}$. A circular boolean range query is issued with centre at u and radius $r = \frac{dist(u,q)}{x}$ that returns true if and only if there exists a facility in the circle. The boolean range query is conducted on the facility R*-tree as in previous works [159] and u is reported as an answer if it returns false.

Algorithm 3 Filtering

Input: user R*-tree, query q, and $\mathcal{A}$
Output: a list of candidates L_{cnd}

```
1:  L_cnd ← φ
2:  insert root of user R*-tree in a stack S
3:  while S is not empty do
4:    retrieve top entry e from S
5:    e' ← PruneEntry(e, A)                          ▷ Algorithm 1
6:    if e' ≠ φ then                                 ▷ e is not pruned
7:      if e is an intermediate or leaf node then
8:        for each child c of e do
9:          if c overlaps with e' then insert c in stack S
10:     else                                         ▷ e is a user
11:       insert e in L_cnd
```

3.5 Experiments

3.5.1 Experimental Setting

To the best of our knowledge, there is no prior algorithm to solve RANN queries. We consider a naïve algorithm (RQ) and make reasonable efforts to devise a significantly improved version of RQ, as explained below.

Range Query (RQ). For each user u, a boolean range query with range $dist(u, q)/x$ is issued on the facility R*-tree (as described in the verification phase above).

Improved Range Query (IRQ). Note that an intermediate or leaf node entry e_u of the user R*-tree cannot contain any RANN if there exists at least one facility f such that $mindist(e_u, q) > x \times maxdist(e_u, f)$, i.e., e_u can be pruned. Based on this, to check whether e_u can be pruned or not, we use a function `isPruned(`e_u`)` that is implemented as follows. The facility R*-tree is traversed in ascending order of $maxdist(e_u, e_f)$ where e_f denotes an entry in the facility R*-tree. The entry e_u is pruned as soon as we find an entry e_f for which $mindist(e_u, q) > x \times maxdist(e_u, e_f)$. To further improve the I/O and CPU cost of `isPruned(`e_u`)`, we do not access the sub-tree of a facility entry e_f if $mindist(e_u, q) < x \times mindist(e_u, e_f)$ because no child of e_f can prune e_u.

The IRQ algorithm is the same as Algorithm 3 except that 1) "**if** `isPruned(`e`)` **then**" replaces lines 5 and 6 of Algorithm 3; and 2) at line 11, the user is reported as an answer instead of inserting it in L_{cnd}. Note that IRQ does not have a pruning and verification phase because it merges all these phases in one algorithm. In our experiments, we observed that the performance of IRQ can be further improved if `isPruned(`e_u`)` is only applied to leaf entries of the user R*-tree. This is because the intermediate nodes are highly unlikely to be pruned and result in un-necessary I/O.

We included this optimization in IRQ.

All algorithms were implemented in C++ and experiments were run on Intel Core $I5$ 2.3GHz PC with 8GB memory running on Debian Linux. Experimental settings are quite similar to the existing work [3]. Specifically, we use the same real data sets containing $175,812$ points from North America (called NA data set hereafter), 2.6 million points from Los Angeles (LA) and 25.8 million points from California (CA). We also generate several synthetic data sets containing $1,000$ to 20 million points following normal distributions. The default real data set is LA containing 2.6 million points. Unless mentioned otherwise, each data set is randomly divided into two sets of almost equal size, one corresponding to the facilities and the other to the users. The page size of each R*-Tree [161] is set to $4,096$ Bytes. We randomly select 100 points from the facility data set and treat them as query points. The cost reported in the experiments correspond to the average cost of a single RANN query. We vary the value of x from 1.1 to 4 and the default value is 1.5.

3.5.2 Experiment Result

3.5.2.1 Effect of buffers

All three algorithms need to traverse facility R*-tree every time a boolean range query is issued to verify a candidate user. Hence, the buffers may reduce the I/O cost. We study the effect of the number of buffers on each algorithm. Each buffer page can hold one node of the R*-tree and we use random eviction strategy. In Fig. 3.10, we report the I/O cost of each algorithm on LA data set for different number of buffers. As expected, the I/O cost of each algorithm decreases with the increase in number of buffers. Note that IRQ is up to two orders of magnitude better than RQ and our algorithm is up to three orders of magnitude better than IRQ. Similar to [3], we use 100 buffer pages for each algorithm in the rest of the experiments.

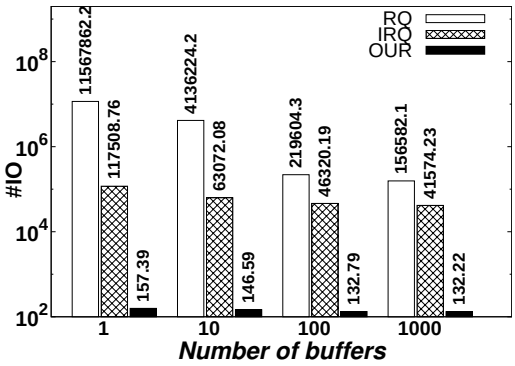
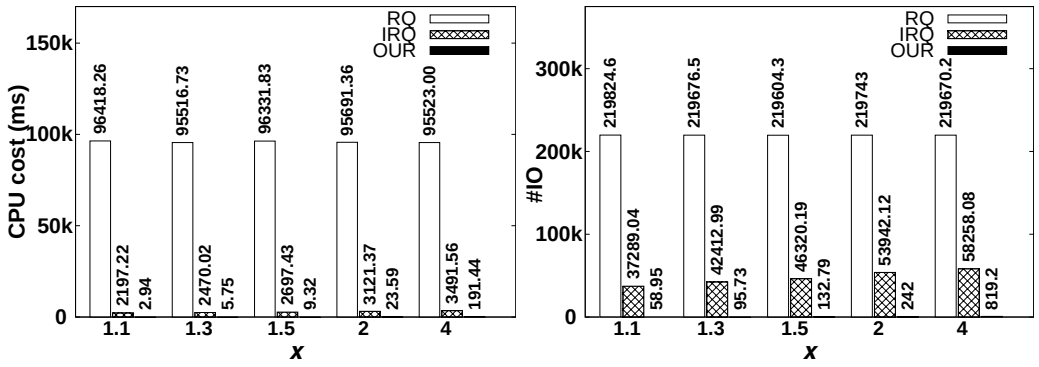


Figure 3.10: Effect of buffers

3.5.2.2 Effect of the x factor

In Fig. 3.11, we study the effect of the x factor on the three algorithms. Specifically, Fig. 3.11(a) shows the CPU cost and Fig. 3.11(b) shows the I/O cost of the three algorithms for varying values of x. In terms of both CPU and I/O cost, our algorithm is up to three orders of magnitude better than IRQ and up to four orders of magnitude better than RQ. The cost of our algorithm and IRQ is higher for larger x factor because the pruning area shrinks as the x factor increases which results in a larger number of candidates and RANNs. Note that the cost of RQ is not significantly affected by the x factor mainly because it needs to verify every user regardless of the value of x.



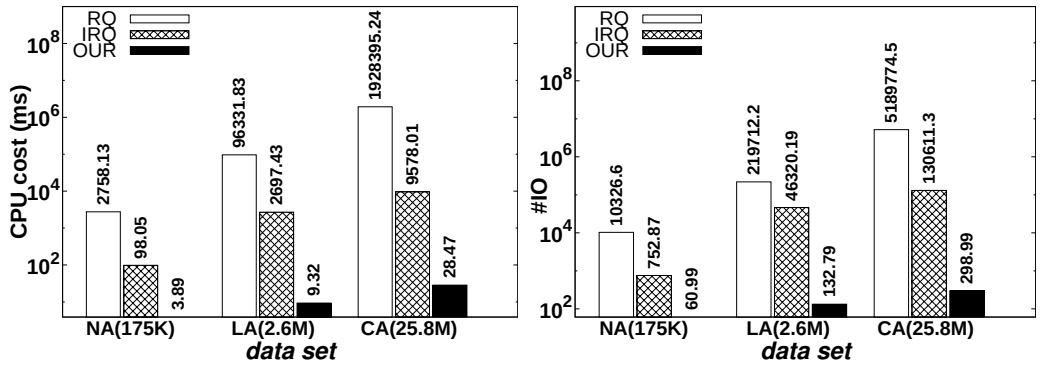
(a) CPU cost (b) I/O cost

Figure 3.11: Effect of the x factor (LA data set)

3.5.2.3 Effect of the data set size

In Fig. 3.12(a) and 3.12(b), we study the effect of data set size on the performance of the three algorithms. Specifically, we conduct experiments on three real data sets: NA (175,000 points), LA (2.6 million points) and CA (25.8 million points). Our algorithm outperforms the other two algorithms and the gap between the three algorithms increases as the data set size increases (please note that log-scale is used in both figures). For example, Fig. 3.12(a) shows that our algorithm is around 25 times faster than IRQ on NA data set and 330 times faster on CA data set. Similarly, Fig. 3.12(b) shows that the I/O cost of our algorithm is around 12 times lower than IRQ for NA data set and almost 430 times lower for CA data set. Also, as expected the cost of each algorithm increases as the data set size increases. This is mainly because the size of each R*-tree increases and more entries are required to be processed.

Since our algorithm is up to several orders of magnitude better than the other algorithms, in the rest of the experiments, we focus on analysing the behavior of our



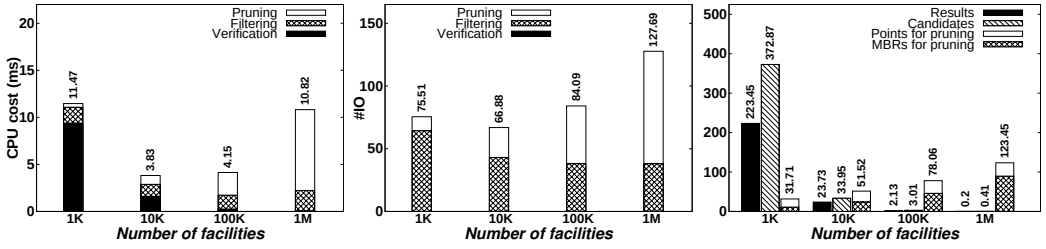
(a) CPU cost (b) I/O cost

Figure 3.12: Performance comparison on different real data sets

algorithm and omit the cost of the other algorithms for better illustration.

3.5.2.4 Effect of relative data size

In the previous experiments, each data set contained almost the same number of users and facilities. Next, we analyse the performance of our algorithm where the number of users and the number of facilities are different. Specifically, in Fig. 3.13 we vary the number of facilities from 1000 to 1 million and the number of users is fixed to $100K$. The sets of facilities and users are generated using normal distribution. Fig. 3.13(a) and Fig. 3.13(b) show the CPU and I/O cost of our algorithm, respectively. Fig. 3.13(c) shows the number of candidates, number of RANNs and the number of entries (facility points and MBRs) used for pruning.



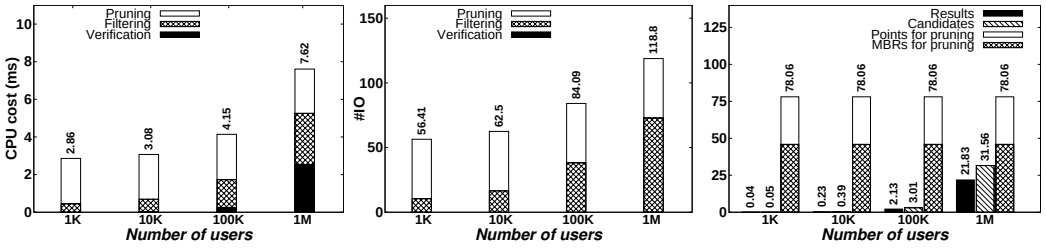
(a) CPU cost (b) I/O cost (c) Stats

Figure 3.13: Effect of varying the number of facilities ($100K$ users)

Fig. 3.13(a) shows that the CPU cost of our algorithm is larger if the number of facilities is too small or too large as compared to the number of users. The reason is as follows. When the number of facilities is too small (e.g., $1,000$), the total area that can be pruned is smaller due to the lower density of the facilities. This results in a larger number of candidates and RANNs (as shown in Fig. 3.13(c)). Hence, the verification cost of the algorithm is larger as shown in Fig. 3.13(a). On the other hand, when the number of facilities is too large (e.g., 1 million), the pruning phase is

the dominant cost of the algorithm. This is because the algorithm needs to access a larger number of entries to prune the search space (see Fig. 3.13(c)).

Fig. 3.13(b) shows the I/O cost of our algorithm. When the number of facilities is too small, the I/O cost of the filtering phase is larger because the area that can be pruned is smaller due to the lower density of facilities data set. The I/O cost of pruning phase increases as the number of facilities increases. This is because the size of facility R*-tree increases and more entries are required to be accessed to prune the search space.



(a) CPU cost (b) I/O cost (c) Entries

Figure 3.14: Effect of varying the number of users ($100K$ facilities)

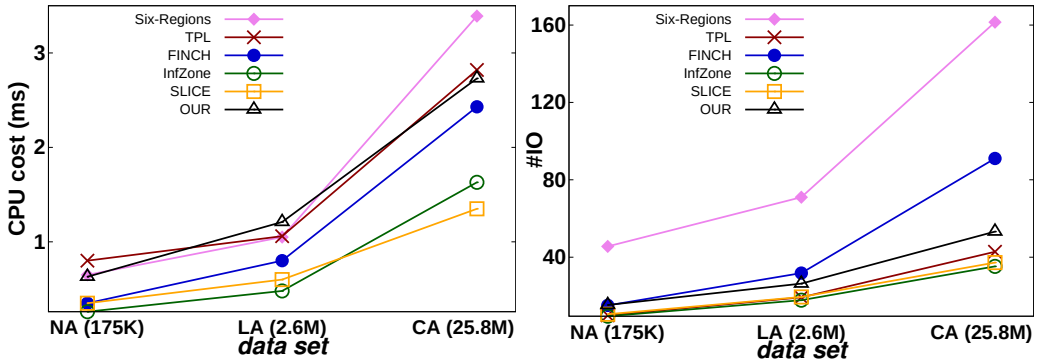
In Fig. 3.14, we vary the number of users from $1,000$ to 1 million and fix the number of facilities to $100K$. Fig. 3.14(a) shows that the CPU cost of the algorithm increases as the number of users increases. This is because the filtering and verification cost of the algorithm increases for larger set of users, e.g., the number of candidate users and RANNs increases (as shown in Fig. 3.14(c)). Similarly, Fig. 3.14(b) shows that the I/O cost of the algorithm also increases for larger number of users. This is because the filtering requires traversing a larger user R*-tree which results in requiring to access more nodes of the users.

Fig. 3.14(c) also shows the effectiveness of the proposed pruning techniques. Note that the number of candidates is much smaller as compared to the total number of users. Furthermore, almost 65% of the candidates are the reverse approximate nearest neighbors. We remark that the verification I/O cost of our algorithm is negligible mainly because most of the nodes accessed during verification are already present in the buffer (from pruning phase or the previously issued boolean range queries).

3.5.2.5 Efficiency compared with RNN algorithms

As stated earlier, there is no previous algorithm to solve RANN queries and the existing algorithms to solve RNN queries cannot be trivially extended. Although we made significant efforts to devise the second competitor IRQ, our algorithm is up to three orders of magnitude better than it. In the absence of a well-known competitor,

readers may find it harder to evaluate the efficiency of an algorithm. Therefore, we compare our algorithm with the most well-known RNN algorithms, namely SLICE [6], InfZone [5], TPL [4], FINCH [114] and six-regions [112]. For our algorithm, we set $x = 1 + 10^{-6}$ because we note that the results of an RANN query is the same as those of an RNN query if x is very close to 1.



(a) CPU cost (b) IO cost

Figure 3.15: Comparison with state-of-the-art RNN algorithms

Fig. 3.15 shows that the performance of our algorithm is comparable to the most popular RNN algorithms which shows the effectiveness of the techniques proposed in this chapter. We remark that this experiment is conducted only to demonstrate that our algorithm is efficient and *it should not be used to draw any conclusion regarding the superiority of our algorithm over any other algorithm and vice versa.* This is because our algorithm solves an inherently different and arguably more challenging problem.

3.6 Conclusions

In this chapter, we propose a variant of RNN queries called reverse approximate nearest neighbors (RANN) queries. An RANN query relaxes the definition of influence using the relative distances between the users and the facilities. RANN queries are motivated by our observation that RkNN queries may be unable to properly capture the notion of influence. We propose an efficient algorithm based on several efficient and effective pruning techniques and non-trivial observations. The pruning techniques are proved to be tight. The extensive experimental study demonstrates that our algorithm is several orders of magnitude better than the competitors.

Chapter 4

Continuous Monitoring of RANN

This chapter presents an efficient algorithm to continuously monitor the RANN queries. We study the scenario when the facility queries are static and the users are continuously moving.

4.1 Overview

In a continuous RANN query, the facilities (e.g., fuel stations) do not change their locations but the users (e.g., drivers) are continuously moving. In such scenario, the results are to be continuously monitored and reported to the query facility. For instance, a fuel station owner may want to continuously monitor the cars influenced by it and may send them promotions. Given a set of facilities F, a set of users U, a set of queries Q and a value of $x > 1$, the problem of continuous monitoring of RANN queries is to continuously monitor the RANNs of every $q \in Q$ when one or more users change their locations.

We assume a client-server paradigm. The server maintains the locations of facilities and the moving users. When a client issues a query, the server computes and sends its initial results to the client. The server also assigns each moving user a *safe zone* which is an area such that the user's movement within this area does not affect the results of any query in the system. The user reports its location to the server only when it leaves its respective safe zone. In this case, the server updates the results of affected queries and sends the relevant clients the updated results. Then, the server computes the new safe zone and sends it to the user. The system also maintains up-to-date results when queries and/or users are added to or deleted from the system. Like most of the existing work on continuous queries [118], we assume a timestamp model in which the server receives the location updates at each

timestamp (e.g., after every t time units) and updates the results accordingly.

We summarize our contributions as follows.

- We propose a novel Voronoi-based algorithm to efficiently monitor concurrent continuous RANN queries for moving objects. A by-product of our techniques for continuous RANN queries is an alternative Voronoi-based algorithm to solve snapshot RANN queries. This Voronoi-based algorithm outperforms our previous algorithm [11] by up to 20 times. To be fair with our previous algorithm, the new Voronoi-based algorithm requires a pre-computed Voronoi diagram and some changes to the standard indexes (e.g., R*-tree, Quadtree). In contrast, our previous algorithm (Section 3.3) can be applied on any branch-and-bound index without requiring any modification. Therefore, the previous algorithm may be preferred by a system administrator who does not want to modify the existing indexes.

- we present a non-trivial extension of the state-of-the-art RkNN monitoring algorithm [5] to handle RANN queries and use it as a competitor for our Voronoi-based algorithm.

- We conduct an extensive experimental study on both real and synthetic data sets to show the effectiveness of our proposed techniques. Our experiments show that our proposed algorithm significantly outperforms the state-of-the-art algorithm in terms of both initial computation cost and continuous monitoring cost

4.2 Proposed Framework

Recall that a user u is an RANN of a query q if and only if it lies outside the pruning circle of every facility (Lemma 3.1 and Lemma 3.2). Thus, a straightforward approach to verify whether a user u is an RANN of a query q is to check pruning circles of all facilities with respect to q and determine if u is outside all these circles or not. Consider a query facility q and two other facilities f_1 and f_2 in Fig. 4.1. C_{f_1} (resp. C_{f_2}) is the pruning circle of facility f_1 (resp. f_2) with respect to the query q. In this example, u_2 is an RANN of q as it is outside all pruning circles. On the other hand, u_1 is not an RANN of q since there is at least one pruning circle (C_{f_2}) that contains it. This simple approach requires $O(|F|)$ to check whether a given user u is an RANN of a query q where $|F|$ is the total number of facilities.

Next, we present an observation that allows checking whether a user is an RANN of a query or not by considering only one pruning circle.

Lemma 4.1 *Let f be the nearest facility of a user u. u is an RANN of a query q if and only if u is outside the pruning circle C_f of f.*

Proof If u is inside the pruning circle C_f, it cannot be an RANN of q (Lemma 3.1), e.g u_1 in Fig. 4.1. Next, we show that if u is outside of C_f, it is guaranteed to be outside of the pruning circle of every other facility f' and, therefore, is an RANN of q. Since f is the nearest facility to u, $dist(u, f) \leq dist(u, f')$ for every other facility f'. Since u lies outside C_f (i.e., $dist(u, q) < x \times dist(u, f)$), we have $dist(u, q) < x \times dist(u, f')$. Thus, u lies outside the pruning circle of f'. Hence if u is not pruned by its nearest facility f then it cannot be pruned by any other facility f'.

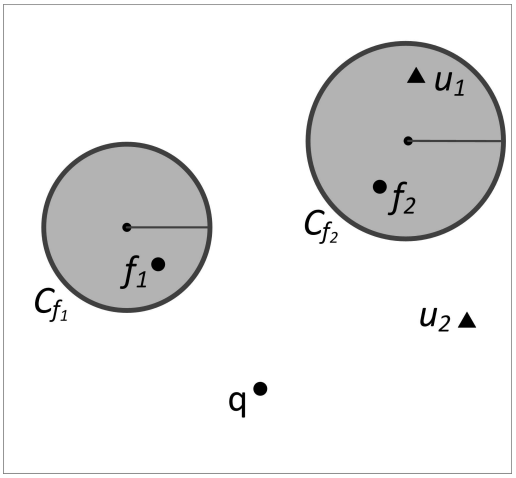


Figure 4.1: RANN verification

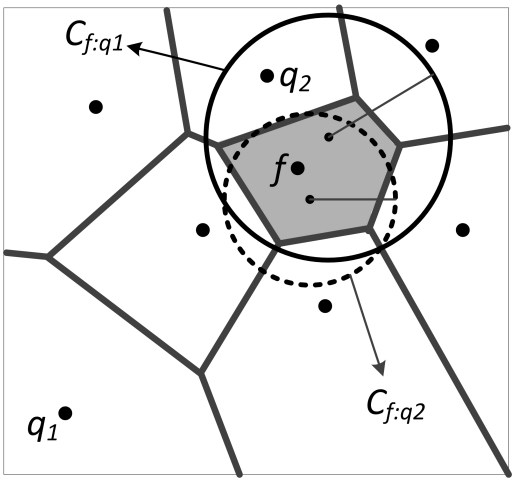


Figure 4.2: Significant facility

Now, consider a case where the system has $|Q|$ queries. A simple approach is to verify the user u using the above lemma for each of the $|Q|$ queries. Assuming that the nearest facility f is known, it requires $O(|Q|)$ to check which queries have u as their RANNs. Next, we present an observation to reduce the number of queries that need to be considered to check which queries have u as their RANNs.

First, we extend the notation to identify the pruning circle of a facility w.r.t. a query. Given a query q and a facility f, we use $C_{f:q}$ to denote the pruning circle of f with respect to query q. If the query is clear by context, we simply use C_f (as we did earlier).

Assume that the Voronoi diagram of all facility points has been computed. Let V_f denote the Voronoi cell of a facility f (e.g., the shaded Voronoi cell in Fig. 4.2). A facility f is called an insignificant facility for a query q if $C_{f:q}$ completely contains its Voronoi cell V_f. On the other hand, a facility f is called a significant facility for q if $C_{f:q}$ does not completely contain V_f. In the example of Fig. 4.2, the facility f is an insignificant facility for q_1 because the pruning circle $C_{f:q_1}$ (the solid circle)

completely contains V_f. On the other hand, f is a significant facility for q_2 because $C_{f:q_2}$ (the dotted circle) does not completely contain V_f. The next lemma shows that a user u in a Voronoi cell V_f can only be an RANN of a query for which f is a significant facility.

Lemma 4.2 *Let u be a user that lies in the Voronoi cell V_f of a facility f. u cannot be an RANN of any query q for which f is an insignificant facility.*

Proof Since u lies in the Voronoi cell V_f, f is the nearest facility to u. Furthermore, since f is an insignificant facility for q (i.e., $C_{f:q}$ completely contains V_f), u is inside $C_{f:q}$. Therefore, u cannot be an RANN of q because it is contained in the pruning circle of its nearest facility (see Lemma 4.1).

We use Lemma 4.1 and Lemma 4.2 to efficiently update RANNs of the queries. Specifically, for each Voronoi cell V_f, we create a list called *sigList* that contains every query q for which f is a significant facility. A user u that moves in a cell V_f can be an RANN of only the queries in the *sigList* of V_f. Hence, we only need $O|K|$ instead of $O|Q|$ to determine what queries have u as its RANN where $|K|$ is the number of queries in *sigList* of V_f. Our experiments show that $|K|$ is significantly smaller than $|Q|$ (around 1% of $|Q|$). In the example of Fig. 4.2, *sigList* of V_f contains only q_2 and a user lying in V_f can be an RANN of only q_2.

To efficiently implement the above observations, we need techniques to efficiently determine the significant facilities for a given query q. Specifically, when a new query q is registered at the system, the algorithm must be able to efficiently determine each of its significant facilities and add q to its *sigList*. The next section provides the details on how to do this efficiently.

4.3 Efficiently Identifying Significant Facilities

A naïve approach is to access each facility $f \in F$, construct its pruning circle $C_{f:q}$ and check whether it contains the Voronoi cell V_f or not. However, this approach is not only computationally expensive but also incurs high I/O cost because the whole facility R*-tree (F-tree) needs to be accessed for each query. We observe that some nodes in the F-tree may not contain any significant facility and can be pruned. Therefore, an efficient approach is to iteratively access F-tree starting from the root node and accessing only the nodes that may contain some significant facilities. Before we present techniques to determine whether a node of F-tree may contain a significant facility or not, we first need to formalize how to check if a facility is significant for q or not.

Consider the example of Fig. 4.3 where V_f is represented as a set of vertices v_1 to v_5. The maximum distance between f and V_f is $dist(f, v_5)$ and the minimum distance

between f and its pruning circle C_f is $dist(f, Z)$. Since $dist(f, Z) > dist(f, v_5)$, we can confirm that$C_f$ fully contains V_f. Therefore, to check if f is a significant facility for q or not (i.e., C_f contains V_f or not), we need to compute the maximum distance between f and V_f (denoted as $maxdist(f, V_f)$) and the minimum distance between f and C_f (denoted as $mindist(f, C_f)$). $maxdist(f, V_f)$ can be easily computed using $dist(f, v_i)$ for each vertex v_i of the Voronoi cell V_f. Formally, $maxdist(f, V_f) = \max_{v_i \in V_f} dist(f, v_i)$. The next lemma shows that $mindist(f, C_f) = \frac{dist(q,f)}{x+1}$.

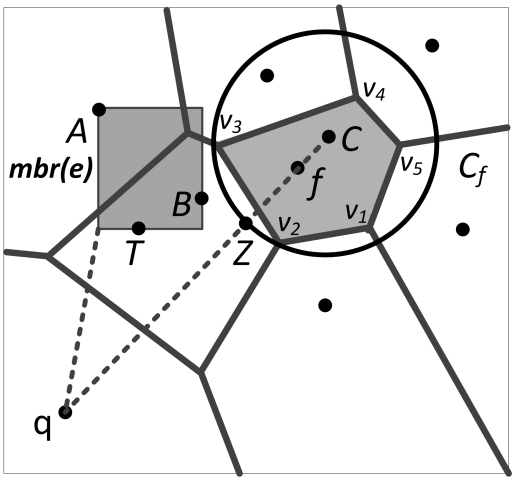


Figure 4.3: Lemma 4.3

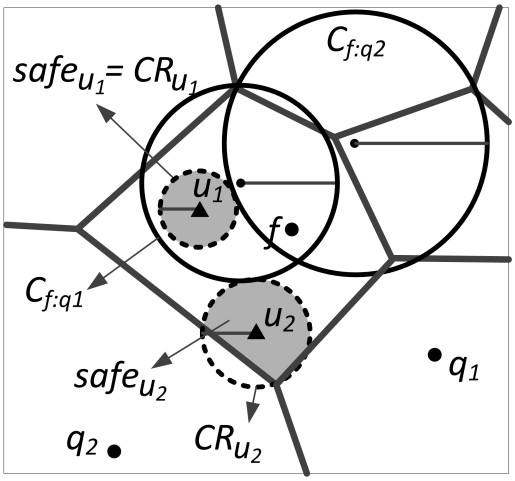


Figure 4.4: Safe zone

Lemma 4.3 *Given a query q, a multiplication factor $x > 1$, and a facility point f, $mindist(f, C_f) = \frac{dist(q,f)}{x+1}$.*

Proof Consider the example of Fig. 4.3. Note that $mindist(f, C_f) = dist(f, Z) = dist(c, Z) - dist(c, f)$. Note that $dist(c, f) = dist(q, c) - dist(q, f)$. By definition of the pruning circle C_f (see Definition 3.2), $dist(q, c) = \frac{x^2 \cdot dist(q,f)}{x^2-1}$. Thus, $dist(c, f) = \frac{x^2 \cdot dist(q,f)}{x^2-1} - dist(q, f) = \frac{dist(q,f)}{x^2-1}$. Since $dist(c, Z)$ is the radius of the pruning circle C_f, by definition of pruning circle, $dist(c, Z) = \frac{x \cdot dist(q,f)}{x^2-1}$. Therefore, $mindist(f, C_f) = \frac{x \cdot dist(q,f)}{x^2-1} - \frac{dist(q,f)}{x^2-1} = \frac{dist(q,f)}{x+1}$.

The next lemma summarizes the above observation. The proof is obvious and is omitted.

Lemma 4.4 *Given a query q and a facility point f, f is an insignificant facility of q if $\frac{dist(q,f)}{x+1} > maxdist(f, V_f)$.*

Now, we are ready to extend the above lemma for a node of the facility R*-tree. Given a leaf or intermediate node e of the facility R*-tree, we define $MaxMaxDist(e)$ as the maximum of $maxdist(f, V_f)$ for every facility $f \in e$, i.e., $MaxMaxDist(e) =$

$\max_{f \in e} maxdist(f, V_f)$. Consider the node e in Figure 4.3 (the shaded rectangle) which contains 3 facility points A, B and T. Assuming that $maxdist(A, V_A)$, $maxdist(B, V_B)$ and $maxdist(T, V_T)$ are 2, 5 and 3, respectively, then $MaxMaxDist(e) = 5$.

The next lemma extends Lemma 4.4 for a node e of the facility R*-tree.

Lemma 4.5 *A node e of the facility R*-tree cannot contain any significant facility for a query q if $\frac{mindist(q,e)}{x+1} > MaxMaxDist(e)$.*

Proof We prove that every $f \in e$ is an insignificant facility for q. Since $dist(q, f) \geq mindist(q, e)$ and $maxdist(f, V_f) \leq MaxMaxDist(e)$, we have $\frac{dist(f,q)}{x+1} > maxdist(f, V_f)$. Therefore, f is an insignificant facility (see Lemma 4.4).

Note that the Voronoi diagram and $MaxMaxDist(e)$ are query independent and can be computed during pre-processing. Specifically, in the pre-processing phase, we first compute a Voronoi diagram of the facilities and calculate $maxdist(f, V_f)$ for each facility f. Finally, $MaxMaxDist(e)$ for each node e in the facility R*-tree is computed in a bottom-up fashion and stored along with e.

4.4 Algorithms

In this section, we present our algorithms to continuously monitor the RANNs of queries. First, we show how to handle the case when a new query is issued (Section 4.4.1). Then, we present the algorithm to handle the case when a new user arrives (Section 4.4.2). In Section 4.4.3, we show how to handle the case when a query or a user is deleted. Finally, we explain how to update the result when one or more users change their locations (Section 4.4.4).

Note that the initial results can be computed by first adding all the queries one by one (Section 4.4.1) and then adding each of the users (Section 4.4.2).

4.4.1 Adding a query

When a new query q is issued, we need to compute its initial results and to insert q into the *sigList* of each *significant facility* of q. Algorithm 4 provides the details. The algorithm initializes a list L with the root of the facility R*-tree (F-tree) (line 1). The entries in the list L are iteratively accessed and are processed as follows. If the accessed entry e is an intermediate or leaf node and cannot be pruned using Lemma 4.5 (line 5), its children are inserted in the list L (line 7).

If e is a facility point, we check whether it is a significant facility or not by checking if V_e is contained by C_e or not (line 8). If e is a significant facility of q, q is inserted in the *sigList* of the Vornoi cell of e (line 9). Then, for each user u in the

Algorithm 4 addQuery(q)

```
1: insert root of F-tree in a list L
2: while L is not empty do
3:    remove an entry e
4:    if e is an intermediate or leaf node then
5:       if e is not pruned then                          ▷ apply Lemma 4.5
6:          insert all children of e in the list L
7:    else                                                ▷ e is a facility
8:       if C_{e:q} does not contain V_e then
9:          insert q in sigList of e
10:         for each user u in Voronoi cell of e do
11:            if u is outside C_{e:q} then
12:               insert u as RANN of q                   ▷ Lemma 4.1
13:               insert q in qList of u
```

Voronoi cell of e, the algorithm checks if u is inside the pruning circle $C_{e:q}$ or not (line 10-12). If u is outside $C_{e:q}$, it will be inserted as an RANN of q and q will be inserted in $qList$ of u (line 13). $qList$ of a user u stores all queries for which u is an RANN. We need the $qList$ to enable us to efficiently find the queries for which u is a result (see Section 4.4.3). The algorithm stops when the list L becomes empty.

Note that at line 10 we need to obtain all users lying in a particular Voronoi cell. To do this, the server maintains the location of each user and the Voronoi cell in which it resides. For each Voronoi cell, the server also maintains a list of users residing in this cell. This list can be used to efficiently find all users in a particular Voronoi cell.

4.4.2 Adding a user

When a new user u arrives, we need to compute the safe zone of u and update the result set of queries for which u is an RANN. Algorithm 5 provides the details. The safe zone of u is a region such that if u is inside this region, the RANN of all queries in the system remain unchanged. Before we present the details of Algorithm 5, we provide an observation to construct the safe zone.

Recall that a user u can be an RANN of only the queries in the $sigList$ of f where f is the nearest facility of u (Lemma 4.2). Furthermore, u is a result of only a query $q_i \in sigList$ of f for which u is outside $C_{f:q_i}$ (Lemma 4.1). In other words, the user u does not affect the results of any query $q_i \in sigList$ as long as it does not leave or enter the circle $C_{f:q_i}$. Therefore, we consider $C_{f:q_i}$ for every q_i in $sigList$ of f and obtain the smallest circle such that as long as u is inside it, it does not leave or enter any of $C_{f:q_i}$. Below, we formally describe this idea.

We use $mindist(u, C_{f:q_i})$ to denote the minimum distance between u and a pruning circle $C_{f:q_i}$ where f is the nearest facility to u. We compute $mindist(u, C_{f:q_i})$ for every

query q_i in the *sigList* of f. The minimum value of $mindist(u, C_{f:q_i})$ is maintained and is stored as $r_{\mathcal{C}}$, i.e., $r_{\mathcal{C}} = \min_{q_i \in sigList_f} mindist(u, C_{f:q_i})$. We create a *critical circle* of u (denoted as CR_u) centered at u with radius $r_{\mathcal{C}}$. Consider the example of user u_1 in Fig. 4.4 that lies in V_f. The *sigList* contains q_1 and q_2 and the pruning circles $C_{f:q_1}$ and $C_{f:q_2}$ are also shown. Since $\min_{q_i \in sigList_f} mindist(u, C_{f:q_i}) = mindist(u_1, C_{f:q_2})$, $r_{\mathcal{C}} = mindist(u_1, C_{f:q_2})$. The shaded circle CR_{u_1} is the *critical circle* for u_1. Note that as long as u_1 is in this circle, it does not enter or leave any pruning circle and, therefore, does not affect the result of q_1 or q_2. Fig. 4.4 also shows the CR_{u_2} for the user u_2.

Note that the construction of CR_u only considers the queries in the *sigList* of facility f where f is the nearest facility to u. Consequently, CR_u is valid as long as u is inside the Voronoi cell V_f of f. If u leaves the Voronoi cell of f, the query results may change. Consider an example of u_2 in Fig. 4.4. If u_2 leaves the Voronoi cell, even though it may still be inside its critical circle CR_{u_2}, the query results may change. Hence, we construct the safe zone of u (denoted as $safe_u$) as the intersection region of CR_u and the Voronoi cell V_f of f, i.e., $safe_u = CR_u \cap V_f$. In Fig 4.4, the safe zones of u_1 and u_2 are the shaded regions.

Now, we are ready to present our algorithm to handle a new user u that arrives. First, Algorithm 5 issues a nearest neighbour query on facility R*-tree to find its nearest facility f (line 1). Then, the Voronoi cell V_f of f is obtained from the pre-computed Voronoi diagram. Subsequently, it accesses the queries in *sigList* of V_f iteratively (line 3). For each query q_i in the *sigList* of f, the algorithm checks whether u is inside $C_{f:q_i}$ or not. If u is outside $C_{f:q_i}$, it is inserted as RANN of q_i and q_i is inserted in *qList* of u (line 4-6). During the iteration, the algorithm maintains $r_{\mathcal{C}}$ (line 7-8) which implies that $r_{\mathcal{C}}$ corresponds to the radius of the critical circle when every q_i has been accessed. The safe zone of u ($safe_u$) is the intersection of CR_u and the Voronoi cell of f (line 10).

Algorithm 5 addUser(u)

```
1: get the nearest facility f of user u
2: r_C ← ∞
3: for each q_i ∈ sigList of V_f do
4:    if u is outside C_{f:q_i} then
5:       insert u as RANN of q_i
6:       insert q_i in qList of u
7:    if r_C < mindist(u, C_{f:q_i}) then
8:       r_C ← mindist(u, C_{f:q_i})
9: create critical circle CR_u centered at u with radius r_C
10: safe_u ← CR_u ∩ V_f
```

4.4.3 Deleting a query or a user

Recall that $qList$ of a user u contains every query q for which u is an RANN. When a user u is deleted, we delete u from the result set of every query in its $qList$ and the affected queries are notified of the updated results.

When a query q is deleted, we retrieve every facility f for which q is in its $sigList$ and delete q from the $sigList$. Note that for each query, we can easily maintain a list of its significant facilities, e.g., when a significant facility is identified at line 9 of Algorithm 4. Then, we remove q from the $qList$ of every user u that has q in its $qList$ (i.e., u is an RANN of q). Note that when a query is removed, the safe zone of some users may become larger. However, we decide not to update the safe zone of such users because it may incur unnecessary computation and communication cost becasue the new safe zones need to be computed and sent to the users. The safe zone of those users will be updated when they leave their safe zones. This does not affect the correctness of the algorithm.

4.4.4 Handling the movement of users

Recall that a user's movement does not affect the results of any query as long as the user remains in its respective safe zone. Therefore, the user sends its location to the server only when it leaves its safe zone. In this case, the results can be easily maintained by first deleting the user from the system and then adding the user (as described in the previous sections). However, some simple yet effective optimizations are possible as described below.

Note that Algorithm 5 (where we add a user u) requires the nearest facility to u (line 1). To find it, a simple approach is to issue a nearest neighbour query on the facility R*-tree for every location update by the user. However, it may incur unnecessary I/O cost. To improve this, we first check if u is still inside the same Voronoi cell. If it is, we do not need to compute its nearest facility because the nearest facility and the Voronoi cell that contains u remain unchanged. Otherwise, we handle the update as follows.

Let f_{old} (resp. $V_{f_{old}}$) be the previous nearest facility (resp. Voronoi cell) of u and f_{new} (resp. $V_{f_{new}}$) is the current nearest facility (resp. Voronoi cell) of u. If u is outside $V_{f_{old}}$, we need to find the current nearest facility to u (f_{new}). Note that $dist(u, f_{new}) < dist(u, f_{old})$. Thus, a nearest neighbour query is issued with a simple modification that every entry e in the facility R*-tree is ignored if $mindist(u, e) > dist(u, f_{old})$.

4.5 Experiment

4.5.1 Competitor

To the best of our knowledge, there is no existing algorithm for continuously monitoring RANN queries. Influence zone [5] is the *state-of-the-art* algorithm for continuous monitoring of RkNN queries. We extend the techniques proposed in [5] to continuously monitor RANN queries by extending the notion of influence zone for RANN queries, i.e., an area such that a user is an RANN of a query if and only if the user is inside this area.

Recall that a user u is an RANN of a query q if and only if it lies outside the pruning circle of every facility (Lemma 3.1 and Lemma 3.2). Thus, the influence zone can be defined as the area outside the pruning circles of all facilities. For example, in Fig 3.3, influence zone is the white area and a user u can be an RANN of q if and only if u lies in the white area. Thus, a straightforward approach to compute influence zone of a query q is to consider the pruning circles for every facility. Influence zone of each query $q \in Q$ can be then computed and indexed using a grid (similar to [5]) and the RANNs of all queries can be monitored using the ideas presented in [5] (see Section 2.3).

However, note that the above approach requires computing $|Q| \times |F|$ pruning circles where $|Q|$ and $|F|$ denote the total number of queries and facilities, respectively. This is not only computationally expensive but also requires huge memory to index all circles in the grid. As explained in Section 5.3.1, it is not trivial to reduce the number of pruning circles because unlike RkNN queries, the users that are very far may still be the RANNs. Nevertheless, to optimize the performance, we carefully design a pruning technique that significantly reduces the number of pruning circles. Our experiments show that this reduces the total number of pruning circles by 30% to 65%. Details of the technique is presented in Section 4.5.2 below.

We partition the data space into an $N \times N$ grid structure (N is set to 64 in the experiments as this gives the best overall performance). For each grid cell, we maintain two lists: *q-list* and *c-list*. The *q-list* of a cell c contains queries whose pruning circles do not overlap c. When a user moves into c, it will be immediately inserted as an RANN of each query in the *q-list* of c. *c-list* of a cell c stores each query q for which there exist at least one facility whose pruning circle with respect to q overlaps c. For each q in *c-list* of c, a list $l_{q:c}$ containing facilities whose pruning circle overlaps c is maintained. When a user u moves into c, it is checked against each query q in *c-list* of c. If u is outside the pruning circles of each facility in $l_{q:c}$, it is inserted as an RANN of q.

We also assign a safe zone for each user. Similar to the Voronoi-based algorithm,

for each user o in cell c, we iterate over all pruning circles in *c-list* of c to get the minimum distance between o and circles in *c-list* of c. We set this distance as the radius of safe circle. The safe zone of o is the intersection of the safe circle and the cell c.

4.5.2 Optimization

We improve the extended influence zone algorithm for continuous RANN queries by reducing the number of pruning circles required to construct the influence zone. Suppose that the algorithm has already accessed a set of facilities $F_{accessed}$ and used their pruning circles to prune the search space. Let the pruned area be denoted as $\mathcal{A}$. A facility $f \notin F_{accessed}$ is called a *useless* facility if it does not prune any additional area, i.e., the pruning circle of f is fully contained in $\mathcal{A}$. We propose techniques to identify an entry e of the facility R*-tree (called useless entry) that contains only useless facilities. The algorithm can then create the influence zone by traversing the facility R*-tree (pruning useless entries) and creating the pruning circles for only the facilities that are not useless. First, we present the concept of an *extended rectangle*.

Definition 4.1 (Extended rectangle) *Let C_p denote the pruning circle of a point p with respect to a query q. Let a, b, c and d denote the four corners of an MBR of a facility R*-tree entry e. Its extended rectangle $ABCD$ is the minimum bounding rectangle of the circles C_a, C_b, C_c, and C_d.*

Consider the MBR *abcd* in Fig 4.1. Its extended rectangle $ABCD$ is the minimum bounding rectangle of the pruning circles C_a, C_b, C_c and C_d as shown in Fig. 4.1.

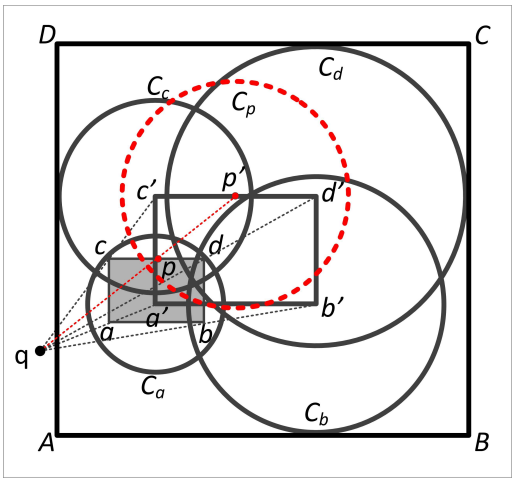


Figure 4.5: Extended rectangle

Lemma 4.6 *Let $\mathcal{A}$ denote the area pruned by $F_{accessed}$. Let abcd be the MBR of an unaccessed facility entry e. The entry e is a useless entry if its extended rectangle ABCD is contained by $\mathcal{A}$.*

Proof Consider MBR *abcd* and its extended rectangle *ABCD* in Figure 4.5. We show that the pruning circle C_p of every point p inside the MBR is contained by $\mathcal{A}$. For a point p, we denote its pruning circle as C_p, the center of the circle as p' and the radius as r_p.

Without loss of generality, let p be a facility point on line cd and C_p be its pruning circle (see the dotted red circle). Since c, d and p are on the same horizontal line, it can be proved that the centers of their pruning circles c', d' and p' are also on a horizontal line. Since $dist(q,p) < dist(q,d)$,according to the definition of pruning circle (Definition 2), $r_p < r_d$. Since $ABCD$ contains C_c and C_d and $r_p < r_d$, r_p can only be outside of $ABCD$ if r_p intersects with the line AD, i.e., r_p can be outside of $ABCD$ only if $r_p > c'p' + r_c$. Next, we prove that this is not possible and $r_p \leq r_c + c'p'$ or $r_c + c'p' - r_p \geq 0$.

From triangle $\triangle qc'p'$, we have

$$c'p'^2 = qc'^2 + qp'^2 - 2.qc'.qp' \cos\theta \tag{4.1}$$

where $\theta = \angle c'qp'$. Since $qc' = \frac{x^2.qc}{x^2-1}$ and $qp' = \frac{x^2.qp}{x^2-1}$ (Definition 2), we have

$$\begin{aligned} c'p'^2 &= (\frac{x^2 \cdot qc}{x^2-1})^2 + (\frac{x^2 \cdot qp}{x^2-1})^2 - 2 \cdot (\frac{x^2 \cdot qc}{x^2-1}) \cdot \\ &\quad (\frac{x^2.qp}{x^2-1}) \cdot cos(\theta) \\ &= \frac{x^4}{(x^2-1)^2}(qc^2 + qp^2 - 2 \cdot qc \cdot qp \cdot cos(\theta)) \end{aligned}$$

Since $qc^2 + qp^2 - 2 \cdot qc \cdot qp \cdot cos(\theta) = cp$, we have

$$c'p' = \sqrt{(\frac{x^4 \cdot cp^2}{(x^2-1)^2})} = \frac{x^2 \cdot cp}{x^2-1} \tag{4.2}$$

Since $r_c = \frac{x \cdot qc}{x^2-1}$, $r_p = \frac{x \cdot qp}{x^2-1}$ (Definition 2) and $c'p' = \frac{x^2.cp}{(x^2-1)}$ (Eq. (4.2)),

$$\begin{aligned} r_c + c'p' - r_p &= \frac{x \cdot qc}{x^2-1} + \frac{x^2 \cdot cp}{(x^2-1)} - \frac{x \cdot qp}{x^2-1} \\ &= \frac{x}{x^2-1}(qc + x.cp - qp) \end{aligned}$$

From triangle inequality, $qc + cp > qp$. Since $x > 1$, $\frac{x}{x^2-1}(qc + x \cdot cp - qp) > 0$ which completes the proof.

4.5.3 Experimental Setting

We compare our algorithm *Voronoi* with the extended influence zone algorithm (denoted as InfZone). We use a real world data set containing point of interests from Los Angeles (LA). The moving objects (i.e., users) are generated by simulating moving cars on the road network of LA using the well-known Brinkhoff data generator [164]. The parameters used in the experiments are shown in Table 6.1 and the default values are shown in bold.

Table 4.1: Experiment Parameters

Parameters	**Range**
x	1.1, **1.5**, 2, 3, 4
Number of facilities (X 1000)	$10, 50, \mathbf{100}, 150$
Number of users (X 1000)	$10, 50, \mathbf{100}, 150$
Users' speed (Km/hr)	40, 60, **80**, 100, 120
Users' Mobility (%)	20, 40, 60, 80, **100**

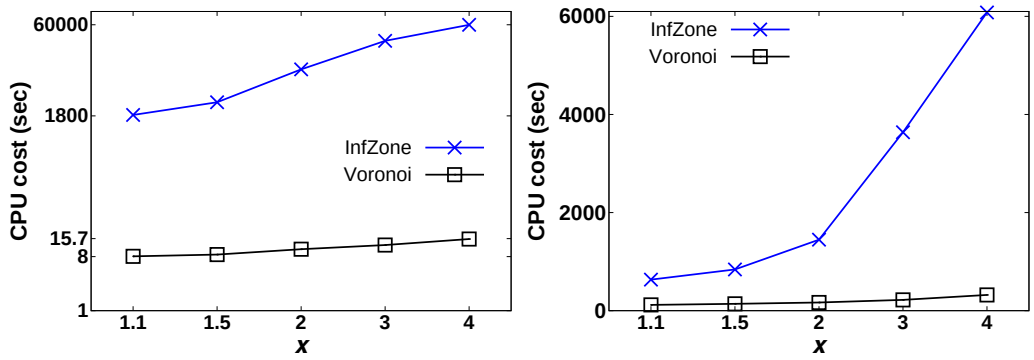
Due to its high memory usage, InfZone cannot handle more than 1000 continuous RANN queries for all data settings. Therefore we use 1000 continuous queries as default. Each query is a randomly selected point from the facility data set and we monitor all 1000 queries for 100 timestamps. We report the total initial cost and the total monitoring cost. The total initial cost is the cost to compute the initial results of all queries. The total monitoring cost is the cost to continuously update the results of the affected queries for 100 timestamps.

4.5.4 Experiment Result

4.5.4.1 Effect of the x factor

Fig. 4.6 studies the effect of the x factor on both algorithms. As expected, the cost of each algorithm increases with the increase of x factor because a smaller area is pruned when x is larger. Fig. 4.6(a) shows that the initial computation cost of *Voronoi* is two to three orders of magnitude lower than that of InfZone and *Voronoi* scales much better (note that log scale is used on y-axis). Fig. 4.6(b) shows that *Voronoi* outperforms InfZone by up to two orders of magnitude and scales much better as the value of x increases. The initial computation cost of *Voronoi* is quite low as shown in the previous section for the snapshot RANN queries. The continuous monitoring cost is also very small due to the effective use of the Voronoi cells and significant

facilities. In contrast, InfZone is significantly more expensive mainly because it requires computing and indexing a large number of pruning circles for each query.

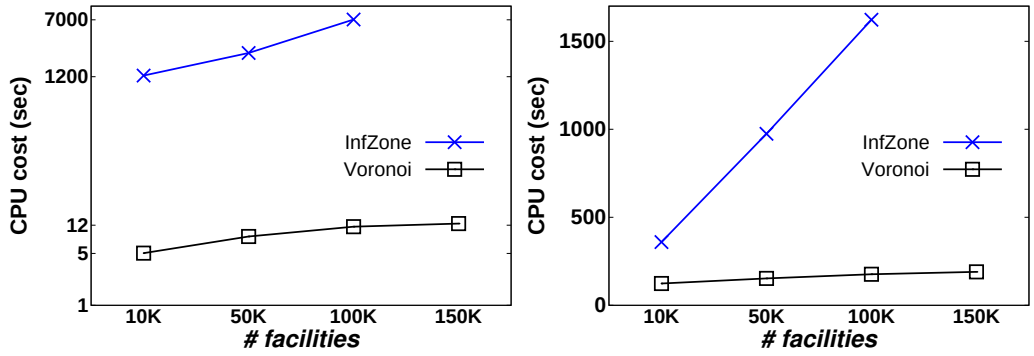


(a) Initial cost (b) Monitoring cost

Figure 4.6: Effect of the x factor

4.5.4.2 Effect of number of facilities

In Fig. 4.7, we study the effect of the number of facilities on both algorithms. InfZone failed to run for $150,000$ facilities because it ran out of memory. The cost of both algorithms increases with the increase in number of facilities mainly because the number of significant facilities and the number of pruning circles increase as the number of facilities increases.



(a) Initial cost (b) Monitoring cost

Figure 4.7: Effect of number of facilities

Fig. 4.7 shows that *Voronoi* significantly outperforms InfZone in terms of both the initial computation cost and the continuous monitoring cost and scales better. In the rest of the experiments, we only compare the monitoring cost of the two algorithms because the initial cost of InfZone is two to three orders of magnitude higher than *Voronoi* for all data settings.

4.5.4.3 Effect of number of users

Fig. 4.8 shows the effect of number of users on the monitoring cost of both algorithms. As expected, the monitoring cost of both algorithms increases with the increase in number of users. *Voronoi* significantly outperforms InfZone and scales much better.

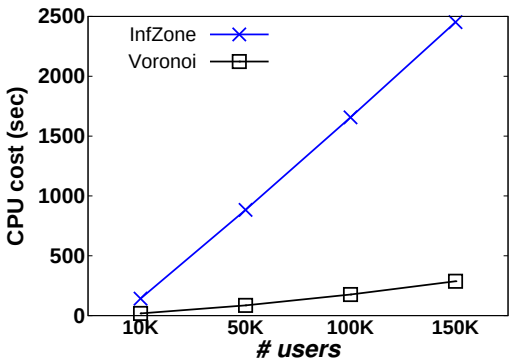


Figure 4.8: Effect of # of users

4.5.4.4 Effect of mobility

Fig. 4.9 studies the effect of mobility which correspond to percentage of the users that move between two timestamps, e.g., 80% mobility corresponds to the data set where 80% of the total users change their locations between two timestamps and the rest of the users are static (e.g., car waiting on traffic light).

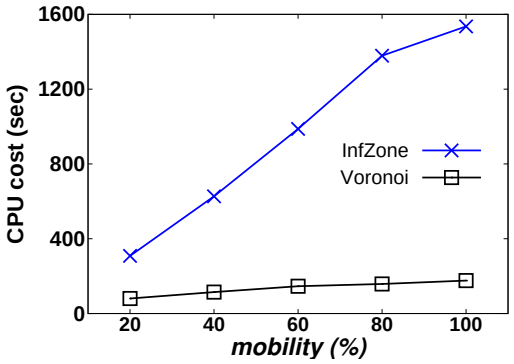
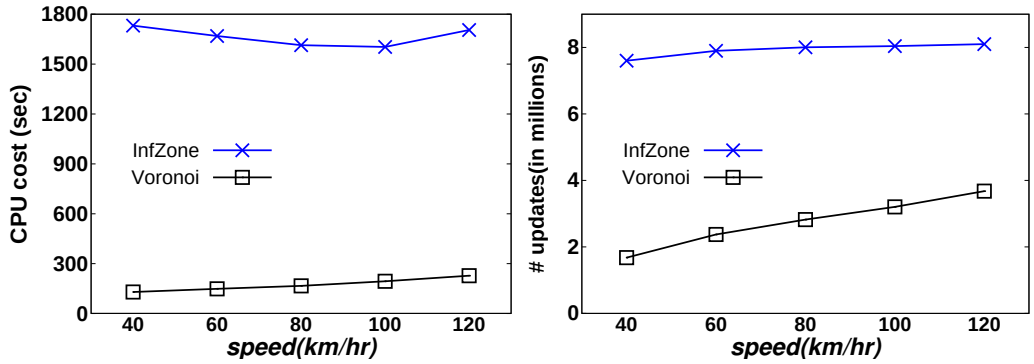


Figure 4.9: Effect of mobility

As expected, the monitoring cost of both algorithms increases with the increase in mobility. *Voronoi* significantly outperforms InfZone and scales much better which is mainly because the safe zones created by *Voronoi* are larger and it takes longer for a user to leave its safe zone which results in requiring fewer updates.

4.5.4.5 Effect of speed

Fig. 4.10 studies the effect of users' speed on the monitoring cost of both algorithms. Fig. 4.10(a) shows that *Voronoi* significantly outperforms InfZone. Fig. 4.10(b) shows the total number of updates in 100 timestamps where an update corresponds to the instance when a user leaves its safe zones. The monitoring cost of *Voronoi* increases with the increase in speed because the number of users that leave their respective safe zones increases with the increase in the speed. Although the number of updates for InfZone also increases with the increase in speed, its monitoring cost is relatively stable. This is because the initial positions of the users generated by Brinkhoff data generator are randomly chosen. Therefore, the initial positions of the users are different in each data set and the trend is difficult to predict because a data sets where more users are located in dense areas will have higher costs.



(a) Monitoring cost (b) Number of updates

Figure 4.10: Effect of the user's speed

4.6 Conclusion

In this chapter, we propose an efficient algorithm to continuously monitor the RANN of queries. We show that using a Voronoi diagram, we only need to check one pruning circle to verify whether a moving user is an RANN of a query. Our algorithm efficiently handles the cases of query and moving user insertion or deletion and the movement of the users. The extensive experimental study on real data sets demonstrate that our algorithms significantly outperform the extended *state-of-the-art* algorithm for RkNN queries.

Chapter 5

Efficient Algorithm for Moving Top-k Queries

In this chapter, we propose an efficient algorithm for continuous monitoring of top-k queries. Unlike range query, nearest neighbor query, reverse nearest neighbor query and reverse approximate nearest neighbor query that only consider the distance between objects and the queries, top-k query considers multiple attributes of the objects. In this chapter, distance to the query point is one of the considered attributes.

5.1 Motivation

Due to the increased use of smart mobile devices, the availability of inexpensive wireless position locators and the more affordable network bandwidth, location based services have gained popularity in the past few years. As a consequence, research on continuous spatial queries to support the location-based services has also attracted significant interest in the past decades. Many algorithms for continuous monitoring of moving queries have been proposed, such as for range query, nearest neighbor query, reverse nearest neighbor query etc.

Each of the above mentioned queries assumes that users, to choose their most important facilities, only consider their distances from the facilities. In many real world applications, distance is not the only criterion considered by the users. In this chapter, we focus on studying the continuous monitoring of spatial queries that involve multiple criterions. Specifically, we study the problem of continuously monitoring the top-k preferred objects for a moving query, where distance between the facilities and the query user is one of the considered criterions.

Consider the example of a driver who is looking for restaurants. He wants restaurants that are close to him, have good reputations and are inexpensive. In this case, the driver considers multiple criteria, i.e., distance, rank and price, to

select his preferred restaurant. The driver may issue a top-k query that considers the restaurants' distances to him, their ranks and food prices to get the k most important restaurants according to his preference. Since the distances between the car and the restaurants change as the car moves, the top-k objects are needed to be updated continuously.

Existing works on continuous top-k queries are designed for a specific scoring function. For example, Wu. et.al., [148] propose an algorithm that uses *weighted distance* which is spatial distance divided by textual relevance. Another example is in [149], which uses *weighted sum* that adds up spatial and non spatial scores. These works are limited in applicability since they only work for a particular scoring function.

There is no single scoring function that is superior for all kinds of problems [165]. To choose an appropriate function, a decision maker should consider characteristics of different scoring functions on different aspects, such as types of data (i.e., qualitative or quantitative), transparency levels, computation complexity and cost [166]. For example, weighted product is appropriate for a multi-dimensional problem, because it eliminates all units of measure [167].

To the best of our knowledge, we are the first to present a solution for continuous top-k queries that works for all monotonic scoring functions, including weighted sum, weighted product, weighted distance etc. Our solution does not utilize properties of a particular scoring function. Instead, it compares the scores to efficiently prune the search space. Hence, it can be applied to any monotonic scoring function.

Below, we summarize our contributions in this chapter.

- To the best of our knowledge, we are the first to propose an algorithm for monitoring of moving top-k queries that can handle any monotonic scoring function, such as weighted sum, weighted product and weighted distance. All existing algorithms [148, 149] are designed for a particular scoring function.
- We develop generic non-trivial pruning techniques that significantly reduce the cost of safe zone computation. Safe zone is a region such that the top-k objects of a query remain unchanged as long as the query lies inside it.
- We conduct extensive experiments on real data set and demonstrate that our algorithm is significantly better than a naïve approach.

5.2 Problem Definition

Let O be a set of objects. In addition to location coordinates, each object has d attributes (dimensions). The i-th attribute value of an object o is denoted as $o[i]$. The distance between a query q and an object o is denoted as $dist(q, o)$ and is

considered as the (d+1)-th dimension of the object, i.e., $o[d+1] = dist(q, o)$. Hence, each object is considered to have $(d+1)$ dimensions. Since $dist(q, o)$ changes with the change in query location, the distance is called the *dynamic* dimension of o. Other attributes of the objects are not affected by the query movement and are called *static* dimensions of the objects.

The static score of an object o (denoted as s_o) is the score computed using the static dimensions according to a given scoring function. Note that the static score of an object does not depend on the distance between o and q. It is called static score because its value remains the same even though the query changes its location. The static score of an object can be computed using any monotonic scoring function.

Consider any monotonic scoring function W, the score of an object is computed using W and the top-k objects with smallest scores are to be returned. Our techniques can be immediately applied to any monotonic scoring function. Some notable examples of monotonic scoring functions used for top-k queries in the past include weighted sum, weighted product and weighted distance. Below, we briefly give examples of some popular monotonic scoring functions.

Weighted Sum [149, 85]. In weighted sum scoring function, a weight w is assigned for each attribute. The weight of i-th attribute is denoted as $w[i]$ where $w[i] \geq 0$ and $\sum_{i=1}^{d+1} w[i] = 1$. Here, $w[d+1]$ is the weight of the dynamic attribute (i.e., $dist(q, o)$) and $w[i]$ (for $1 \leq i \leq d$) is the weight of each static attribute $o[i]$. The score of an object o with regards to the query q is denoted as $score(q, o)$ and is computed using the following function.

$$score(q, o) = w[d+1] \cdot dist(q, o) + \sum_{i=1}^{d} w[i] \cdot o[i] \tag{5.1}$$

Weighted Product [168]. Weighted product is very similar to weighted sum. The difference is that instead of addition, it uses multiplication of attributes to compute the score of an object. The scoring function in weighted product is presented below.

$$score(q, o) = dist(q, o)^{w[d+1]} \cdot \prod_{i=1}^{d} o[i]^{w[i]} \tag{5.2}$$

Weighted Distance [148, 169]. In weighted distance, the score of dynamic attribute is divided by the score of static attributes (s_o). Weighted distance has been used in many works on geotextual queries, where the static score corresponds to the textual relevance.

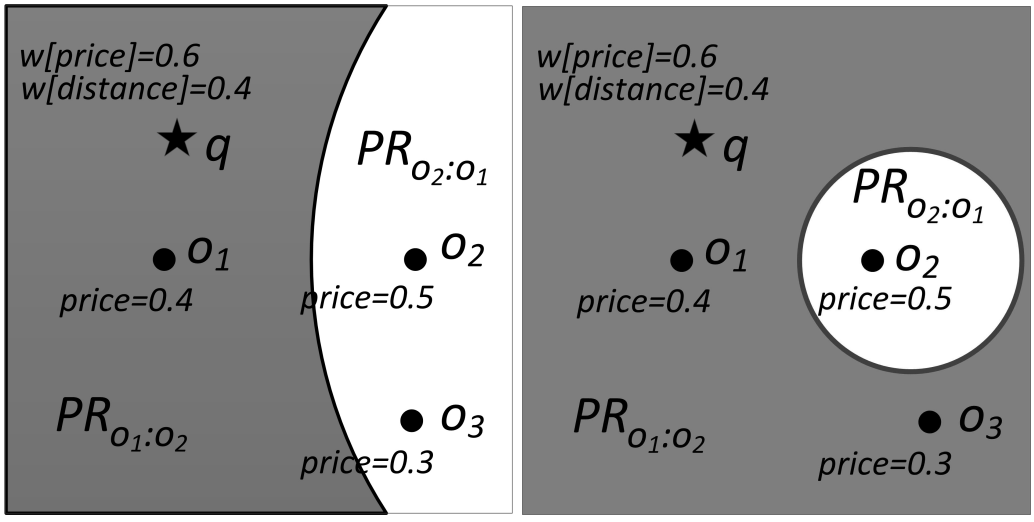
$$score(q, o) = \frac{dist(q, o)}{s_o} \tag{5.3}$$

A top-k query returns the k objects with lowest scores according to a given

monotonic scoring function W. The dynamic attribute of the object, i.e., distance between object and the query point, changes as the query q changes its location. Since q is continuously moving, the top-k objects of q need to be continuously updated. In this chapter, we study the problem of continuously monitoring the top-k objects of a moving query.

5.3 Proposed Solution

We propose a safe zone based approach to efficiently monitor the top-k objects of a query. The safe zone (Z) of a query q is a region containing q such that as long as q remains inside Z, its top-k objects as well as their relative order remain unchanged. Consider an example of a query q and three restaurants (o_1 to o_3) in Fig. 5.1. The scoring function and objects' static attributes (i.e. price) are shown in in the figure. Assume a weighted sum method is used, where the weight of price is 0.6 and the weight of distance is 0.4, and the distance between q and o_1, o_2 and o_3 are 0.2, 0.5 and 0.9 respectively. The score of o_1 ($score(q, o_1)$) with respect to the scoring function W is $score(q, o_1) = 0.4 * 0.2 + 0.6 * 0.4 = 0.32$. Similarly, $score(q, o_2)$ and $score(q, o_3)$ are 0.5 and 0.54 respectively. In the case of $k = 2$, the top-k results of q are o_1 and o_2. In our solution, we construct a safe zone for q such that as long as q is inside it, o_1 and o_2 are the top-2 objects for q respectively.



(a) Weighted sum (b) Weighted product

Figure 5.1: Preferred regions

We say that o_1 is more preferred than o_2 as long the score of o_1 with respect to W is better than the score of o_2. Since the static score does not change, the score of an object may change if the dynamic attribute of the object (i.e., distance to the query) changes. In other words, the top-k objects of a query as well as their order,

may change when the query changes its location. Now, we present the definition of *preferred region*.

Definition 5.1 ***Preferred Region.*** *Given two objects o_1, o_2 and a query q with any monotonic scoring function W, the* ***preferred region*** *of o_1 (denoted as $PR_{o_1:o_2}$) is a region such that as long as q is inside $PR_{o_1:o_2}$, o_1 is more preferred than o_2 with respect to W, i.e., $score(q, o_1) < score(q, o_2)$. Similarly, o_2 is more preferred than o_1 if q is inside $PR_{o_2:o_1}$.*

Fig. 5.1 shows the example of preferred regions in different scoring functions. In Fig 5.1(a), a weighted sum scoring function is used. $PR_{o_1:o_2}$ is the shaded area (which is defined by a hyperbola) and $PR_{o_2:o_1}$ is the white area. As long as q is inside the shaded area, o_1 is more preferred than o_2. Similarly, if q is inside the white area, o_2 is more preferred to q than o_1. In Fig. 5.1(b), a weighted product scoring function is used. $PR_{o_1:o_2}$ is the shaded area and $PR_{o_2:o_1}$ is the white area which is defined by an *apollonius circle*.

Note that if o_1 is a query result, the query point q is outside $PR_{o_2:o_1}$. Similarly, given a set of objects O, q is outside $PR_{o_i:o_1}$ for every object $o_i \in O$. If q enters a $PR_{o_i:o_1}$, the query results may change. Hence, the safe zone of q can be defined using the minimum distance between q and $PR_{o_i:o_1}$ for every $o_i \in O$.

5.3.1 Challenges

Given a query q with any monotonic scoring function W, the safe zone of a query q is a region containing q that is defined by all preferred regions. Specifically, it is the intersection of preferred regions defined by all objects in the data space.

Consider the example of a query q in Fig. 5.2. Assume that a weighted sum scoring function is used and o_1 is the query result. The safe zone (Z) of q is the shaded region which is the intersection of the preferred regions defined by o_1 and all objects in the data space. i.e,. $Z = PR_{o_1:o_2} \cap PR_{o_1:o_3}$. As long as q is inside Z, the query results remain unchanged. Once q leaves Z, the query results or the order of objects in the result set may have changed.

Computing the safe zone for a query may be inefficient and even impossible for some complex monotonic scoring functions, since we do not know the shape of the preferred region. The shape of a preferred region is defined by the scoring function. It can be any geographic shape, such as a hyperbola, a circle, or even a complex shape like disjoint circles. Since a top-k query may use different scoring functions, we need a generic technique that is applicable to any geographic shape. We present our safe zone based technique to answer such requirement in the next section.

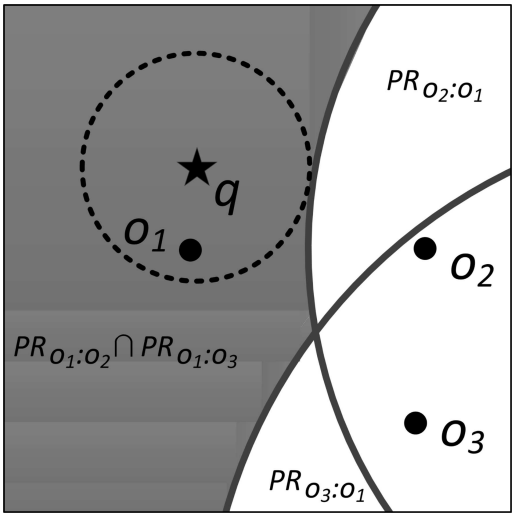


Figure 5.2: Safe zone

5.3.2 Safe zone overview

Given a top-1 query q with a monotonic scoring function W, and assume that o_1 is the query result, q is outside pruning region $PR_{o_i:o_1}$ for every *other* object $o_i \in O$. As long as q is outside these preferred regions, the query results do not change. When the query enters a preferred region $PR_{o_i:o_1}$, the query results may have changed. Hence, the safe zone of q can be defined using the closest preferred region from q.

We propose a circle shaped safe zone with radius of the minimum distance between the query and preferred regions. A circle is easy to compute, and also it does not require high cost to check if an object is inside a circle. In the example in Fig. 5.2, the safe zone of q (Z) is the dashed circle centered at q with the radius r_Z, which is the minimum of the minimum distance between q and $PR_{o_2:o_1}$ and the minimum distance between q and $PR_{o_3:o_1}$, i.e., $r_Z = min\{mindist(q, PR_{o_2:o_1}), mindist(q, PR_{o_3:o_1})\}$.

A naïve approach to compute the radius of the safe zone is to consider all objects in the data space. Assume that for $k = 1$, the query result of a top-k query is object o_1. For every other object $o_i \in O$, we may compute the minimum distance between q and the preferred region of o_1 and o_i ($PR_{o_i:o_1}$). However, computing the minimum distance between q and preferred region $PR_{o_i:o_1}$ is not trivial since the shape of the preferred region may be quite complex depending on the scoring function used. Next, we present our technique to compute the distance between a point and an arbitrary shape of preferred region.

5.3.3 Computing minimum distance between query and preferred region

Intuitively, if we know a very small rectangle that overlaps an unknown shape, the minimum distance between this rectangle and q serves as a lower bound on the minimum distance between q to the part of the shape that is overlapped by the rectangle. If we take the minimum among all these rectangles, we will get the lower bound on the minimum distance between query and the unknown shape. Consider a query q and a pruning region $PR_{o_2:o_1}$ in Fig. 5.3. Rectangles c_1 to c_6 overlap $PR_{o_2:o_1}$. Rectangle that gives the lower bound distance is c_4 (shown in solid line) and hence $mindist(q, PR_{o_2:o_1}) \leq mindist(q, c4)$. Since it is a lower bound, the lower bound on the minimum distance ($LB\_mindist$) between q and $PR_{o_2:o_1}$ is the minimum distance between q and c_4, i.e., $LB\_mindist(q, PR_{o_2:o_1}) = mindist(q, c4)$.

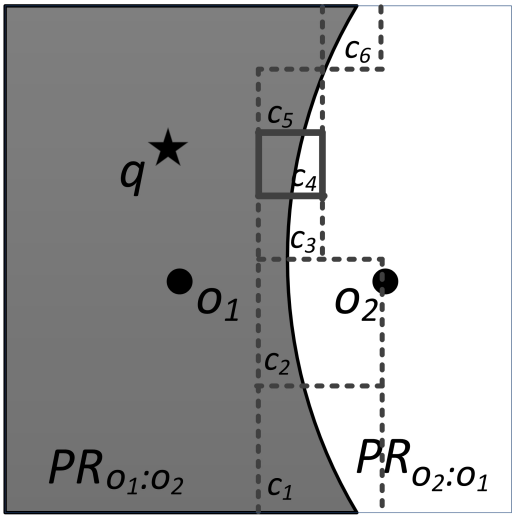


Figure 5.3: Observation

We use this observation to compute the lower bound on the minimum distance between a query point q and a preferred region PR. We iteratively divide the space into smaller rectangles and we ignore the rectangles that do not overlap the shape. We maintain the remaining rectangles based on the minimum distance to q . The process is terminated when the rectangle area is smaller than a parameter ϵ. If we compute the minimum distance between q and R, we get the lower bound of the minimum distance between q and PR.

Now we present the definition of *minimum score* and *maximum score*, as well as Lemma 5.1 to prune non overlapping rectangles.

Definition 5.2 ***Minimum (resp. maximum) score.*** *Given a query q with a scoring function W, an object o and a rectangle R, the minimum (resp. maximum)*

score of o with respect to W and R, denoted as minScore(q,o,R) (resp. maxScore(q,o,R)), is the lowest (resp. highest) possible score of o considering the location of q in R. Since distance(q, o) is the only dynamic attribute, minScore(q,o,R) (resp. maxScore(q,o,R)) is obtained at mindist(o, R) (resp. maxDist(o, R)).

Example Consider two objects o_1 and o_2 and a rectangle R in Fig 5.4. Assume that the scoring function is weighted sum and hence $PR_{o_1:o_2}$ is a hyperbola. $minScore(q, o_1, R)$ is obtained in q, since $mindist(o_1, R) = dist(q, o_1)$, whereas $maxScore(q, o_1, R)$ is obtained when the query point is in q'', since $maxdist(o_1, R) = dist(q'', o_1)$. Similarly, $minScore(q, o_2, R)$ and $maxScore(q, o_2, R)$ are obtained at q' and q''.

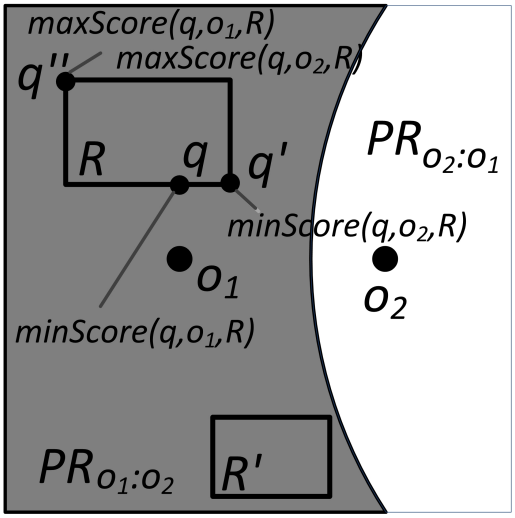


Figure 5.4: minimum & maximum score

Next, we present Lemma 5.1 to efficiently check that a rectangle does not overlap a preferred region.

Lemma 5.1 *Given a rectangle R, two objects o_1 and o_2 and a monotonic scoring function W, R completely lies in $PR_{o_1:o_2}$ if $minScore(q, o_2, R) > maxScore(q, o_1, R)$.*

Proof Assume $minScore(q, o_2, R)$ is obtained at q' and $maxScore(q, o_1, R)$ is achieved at q'', then $mindist(o_2, R) = dist(q', o_2)$ and $maxdist(o_1, R) = dist(q'', o_1)$. Since distance is the only dynamic attribute and W is a monotonic function, $score(q, o_2)$ increases as the increase of $dist(q, o_2)$. Similarly, $score(q, o_1)$ decreases as the decrease of $dist(q, o_1)$. Since $minScore(q, o_2, R) > maxScore(q, o_1, R)$, then for any point q in R, $score(q, o_2) > score(q, o_1)$, i.e., $score(q, o_1)$ will always be smaller than $score(q, o_2)$ where ever the location of q in R is. In other words, R is fully inside $PR_{o_1:o_2}$.

Lemma 5.1 prunes the rectangle that is guaranteed not to overlap the preferred region. On the other side, when that condition is not met (e.g., $minScore(q, o_2, R) \leq maxScore(q, o_1, R)$), we use Lemma 5.2 to further check if R can actually be pruned. Consider rectangle R' in Fig. 5.4 and assume that $minScore(q, o_2, R') < maxScore(q, o_2, R')$, R' can not be pruned using Lemma 5.1 even though it actually does not overlap preferred region $PR_{o_2:o_1}$.

Before we present Lemma 5.2, we present the definition of shared score, shared distance and shared area.

Definition 5.3 ***Shared score.*** *Given a query q, two objects o_1 and o_2, and a rectangle R, the shared score of o_1 and o_2 with respect to q and R, denoted as $sharedScore(o_1, o_2, R)$, is a range of score that is shared by $score(q, o_1)$ and $score(q, o_2)$ considering the location of q in R. The minimum (resp. maximum) shared score of o_1 and o_2, denoted as $minSharedScore(o_1, o_2, R)$ (resp. $maxSharedScore(o_1, o_2, R)$) is the minimum (resp. maximum) value of shared score, such that $minScore(q, o_1, R) \leq minSharedScore(o_1, o_2, R) \leq maxSharedScore(o_1, o_2, R) \leq maxScore(q, o_2, R)$ or $minScore(q, o_2, R) \leq minSharedScore(o_1, o_2, R) \leq maxSharedScore(o_1, o_2, R) \leq maxScore(q, o_1, R)$.*

Definition 5.4 ***Minimum & maximum shared distance.*** *Given a query q, two objects o_1 and o_2, and a rectangle R, the minimum shared distance of o_1, denoted as $minSharedDistance(o_1, o_2, R)$ is the distance between o_1 and a point q inside R such that $score(q, o_1) = minSharedScore(o_1, o_2, R)$. $minSharedDistance(o_1, o_2, R)$ is computed as:*

$$minSharedDistance(o_1, o_2, R) = \frac{minSharedScore(o_1, o_2, R) - s_{o_1}}{w[d+1]}$$

where s_{o_1} is the static score of o_1 and $w[d+1]$ is the weight of dynamic attribute. The maximum shared distance ($maxSharedDistance(o_1, o_2, R)$) is computed in similar way using $maxSharedScore(o_1, o_2, R)$.

If a rectangle R cannot be pruned by Lemma 5.1, there must be a range of score between $minSharedScore(o_1, o_2, R)$ and $maxSharedScore(o_1, o_2, R)$ that is shared by $score(q, o_1)$ and $score(q, o_2)$. Based on this shared score, the values of minimum and maximum shared distance of o_1 and o_2 can be obtained. In Fig 5.5, $minSharedDistance(o_1, o_2, R)$ and $maxSharedDistance(o_1, o_2, R)$ are shown as the radius of circles centered at o_1. Similarly, $minSharedDistance(o_2, o_1, R)$ and $maxSharedDistance(o_2, o_1, R)$ are illustrated as the radius of circles centered at o_2.

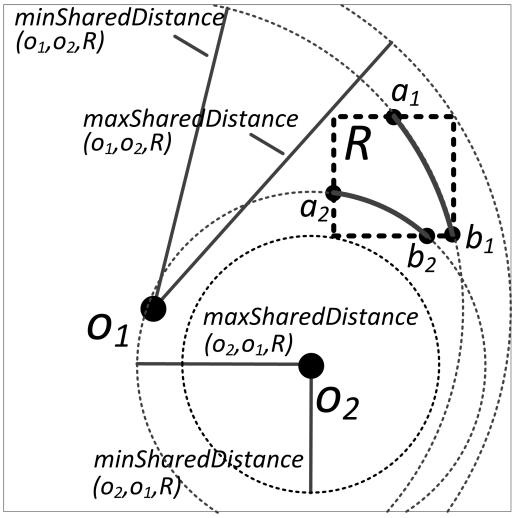


Figure 5.5: Lemma 5.2

Definition 5.5 ***Shared area.*** *Given a query q, two objects o_1 and o_2, a rectangle R, and the minimum and maximum shared distance of o_1 and o_2, the shared area of o_1, denoted as $\mathcal{A}_{o_1}$ is the area between two circles, $c_{o_1}^{min}$ and $c_{o_1}^{max}$ where $c_{o_1}^{min}$ (resp. $c_{o_1}^{max}$) is a circle centered at o_1 with radius $minSharedDistance(o_1, o_2, R)$ (resp. $maxSharedDistance(o_1, o_2, R)$). Specifically $\mathcal{A}_{o_1} = c_{o_1}^{max} - c_{o_1}^{min}$.*

Now we present Lemma 5.2 to further check if a rectangle can be pruned.

Lemma 5.2 *Given a query q, two objects o_1 and o_2 and a rectangle R, R completely lies in $PR_{o_1:o_2}$ if there is no intersection area between $\mathcal{A}_{o_1}$, $\mathcal{A}_{o_2}$ and R, i.e,. $\mathcal{A}_{o_1} \cap \mathcal{A}_{o_2} \cap R = \emptyset$.*

Proof If $\mathcal{A}_{o_1} \cap \mathcal{A}_{o_2} \cap R = \emptyset$, it shows that there is no shared score between $score(q, o_1)$ and $score(q, o_2)$ wherever the location of the query q in R is, i.e,. for any query q in R, $score(q, o_1) < score(q, o_2)$.

In the implementation, we use intersection points of circles c_o^{max} or c_o^{min} and the rectangle R to check if there exists such intersection area. Consider two objects o_1 and o_2 and a rectangle R in Fig. 5.5. a_1 and b_1 are the intersection points of R and $c_{o_1}^{min}$. Similarly, a_2 and b_2 are the intersection points of R and $c_{o_2}^{max}$. If a_1, b_1, a_2 and b_2 are outside R, then R lies completely inside $PR(o_1, o_2)$.

We use Lemma 5.1 and 5.2 in Algorithm 6 to efficiently obtain a lower bound on the minimum distance between the query point (q) and a preferred region PR. First, the rectangle R is set to cover the whole space (line 2) and is inserted into a *heap* sorted based on the minimum distance between q and R. R is divided into smaller rectangles and those that overlap PR are inserted into the *heap* (line 10).

The process continues until the size of the de-heaped entry e is smaller than ϵ (line 6). The minimum distance between q and e is then returned (line 7).

Algorithm 6 $mindist(q, PR_{o_2:o_1})$

```
1:  mindist(q, PR_{o2:o1}) ← ∞
2:  R ← whole space
3:  insert R in a min-heap heap
4:  while heap is not empty do
5:    de-heap an entry e
6:    if area(e) < ϵ then
7:      return mindist(q, PR_{o2:o1}) = mindist(q, e)
8:    for each child c of e do
9:      if R overlaps PR_{o2:o1} then                ▷ lemma 5.1 & 5.2
10:       insert c in heap according to mindist(q, c)
```

Fig. 5.6 illustrates steps to compute the minimum distance between q and a preferred region in the case where a top-1 query returns o_1 assuming a weighted sum scoring function is used. First, the whole data space is divided into four smaller rectangles $c_{1:1}$ to $c_{1:4}$ (we use $c_{1:4}$ to denote rectangle 4 in iteration 1). In Fig. 5.6, $c_{1:3}$ and $c_{1:4}$ are inserted in the *heap* since they overlap $PR_{o_2:o_1}$. $c_{1:1}$ and $c_{1:2}$ are discarded. At this stage, the *heap* is updated to $heap = \{c_{1:4}, c_{1:3}\}$. $c_{1:4}$ is then de-heaped and is divided into smaller rectangles $c_{2:1} - c_{2:4}$. *heap* is updated to $heap = \{c_{2:2}, c_{2:1}, c_{1:3}\}$. In the third iteration, *heap* is updated as follows, $heap = \{c_{3:1}, c_{3:2}, c_{2:1}, c_{1:3}\}$. The iteration stops when the area of the de-heaped entry e is smaller than ϵ such that the minimum distance between q and e reflects the minimum distance between q and $PR_{o_2:o_1}$.

5.3.4 Computing safe zone

As mentioned in the previous section, a straight forward approach to compute the valid radius of the safe zone is to consider the preferred region of all objects in the data space. However, this approach is expensive since for each object o, we compute the minimum distance between the query and the preferred region of o. We propose a technique to reduce the cost of safe zone computation.

We compute the minimum possible score of an object considering the location of the query in the current safe zone and compare it with the maximum possible score of the query result to quickly check if the object can be ignored. Consider an object o_2 and a top-1 query that returns o_1 in Fig. 5.7. The current safe zone (Z) is the circle centered at q with radius r_Z. The minimum possible score of o_2 with regards to q and Z (denoted as $minScore(q, o_2, Z)$) is obtained when the distance

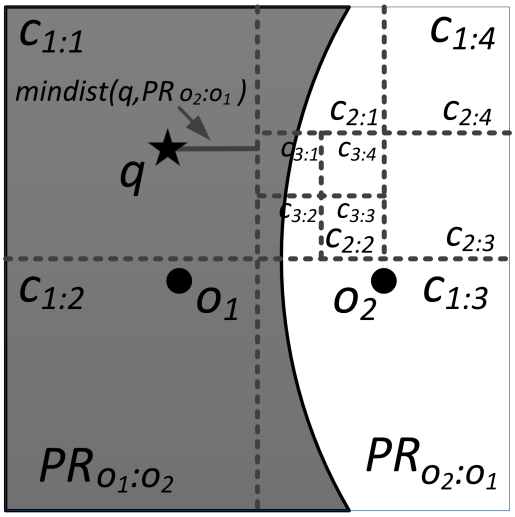


Figure 5.6: $mindist(q, PR_{o_2:o_1})$

between o_2 and q is minimum. Similarly, the maximum possible score of the query result o_1 (denoted as $maxScore(q, o_1, Z)$) is obtained when the distance between q and o_1 is maximum. In Fig. 5.7, $minScore(q, o_2, Z)$ is obtained when the query is in q' whereas $maxScore(q, o_1, Z)$ is obtained when the query is in q''.

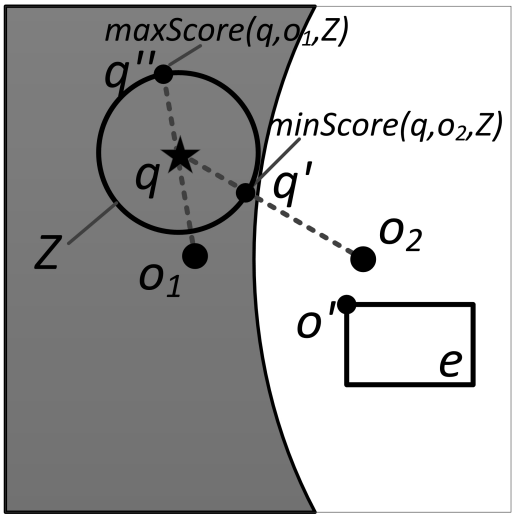


Figure 5.7: Pruning

Now we introduce Lemma 5.3 to check if an object can be quickly pruned.

Lemma 5.3 *Given a query q, its safe zone Z with radius r_Z, and its top-k objects (top-k), any object o' can be pruned, if for every object o in top-k, $maxScore(q, o, Z) < minScore(q, o', Z)$.*

Proof If $maxScore(q, o, Z) < minScore(q, o', Z)$, Z does not overlap $PR_{o':o}$ and

$mindist(q, PR_{o':o}) > r_Z$. Processing o' will not affect Z, hence o' can be safely pruned.

We extend Lemma 5.3 and use the minimum possible score of a node to check if the node can be safely pruned. Consider an object node e and a top-1 query that returns o_1 in Fig. 5.7. $minScore(q, o', Z)$ is the lowest possible score of an object in e, considering the location of q in Z. If $minScore(q, o', Z) > maxScore(q, o_1, R)$, there is no object o' in e such that $PR_{o':o_1}$ overlaps Z and hence e can be safely pruned.

Next, we present our algorithm to continuously monitor the result of top-k queries. For each query q, we assign a safe zone Z such that as long as q is inside Z, the top-k objects of q as well as their relative order remain unchanged. When q leaves Z, both top-k objects and the safe zone of q are recomputed.

Algorithm 7 shows the details of the safe zone computation. Initially, the radius of the safe zone r_Z is set to be infinity and the root of object R-tree is inserted into min-heap (line 1-2). When an intermediate node e is encountered, the algorithm checks if e can be pruned (line 5), otherwise, its children are inserted into min-heap (line 13). When an object e is de-heaped, for each object o in top-k, the minimum distance between the query q and the preferred region $PR_{o:e}$ is computed and r_Z is updated if necessary (line 6-8). The algorithms stops when the heap becomes empty.

Algorithm 7 compute safe zone

```
1:  r_Z ← ∞
2:  insert root of R-tree in a heap H sorted according to minimum score
3:  while H is not empty do
4:    de-heap an entry e
5:    if e is not pruned then                                  ▷ lemma 5.3
6:      if e is not an intermediate or leaf node then
7:        for each object o in top-k do
8:          r_Z = min(r_Z, mindist(q, PR_{o:e}))
9:        if top-k < k then
10:         insert e into top-k
11:     else
12:       for each child c of e do
13:         insert c in H
14: return top-k and Z
```

5.4 Experiment

5.4.1 Experimental Setting

We use weighted sum scoring function in our experiment. We devise a competitor algorithm with an assumption that there exists an oracle that computes our proposed safe zone without incurring any cost. We call such algorithm as pseudo-supreme (**PS**) since we remark that our safe zone is not optimal. In pseudo-supreme algorithm, the objects are indexed with R-tree. The algorithm maintains a min-heap sorted based on the minimum score of the R-tree entries. The algorithm immediately stops when k objects have been retrieved. The oracle calls Algorithm 7 to compute the safe zone in the initial time and every time when the query leaves its safe zone. To better illustrate the performance of our algorithm, we also compare it with a naïve algorithm (**Heap**) that does not apply a safe zone.

We use real data set that contains 175,813 POIs in North America. To investigate the effect of the object cardinality, we select the required number of POI's from the real data set. The objects are indexed in a disk-resident R-tree with the node size is set to 4096 bytes. One hundred queries are generated using Brinkhoff data generator [164] which simulates the moving cars in road network in North America. We compare our algorithm (**Our**) with **PS** and **Heap** in different parameters as shown in Table 6.1 (default values are shown bold). All results reported in Section 5.4.2 are the cost for continuously monitoring all query results for one hour (3600 timestamps).

Table 5.1: Experiment Parameters

Parameters	**Range**
Number of objects (×1000)	1, 10, **100**, 150
k (for moving top-k)	1, **3**, 5, 7
Dimensionality of R-tree	3, **4**, 5, 6

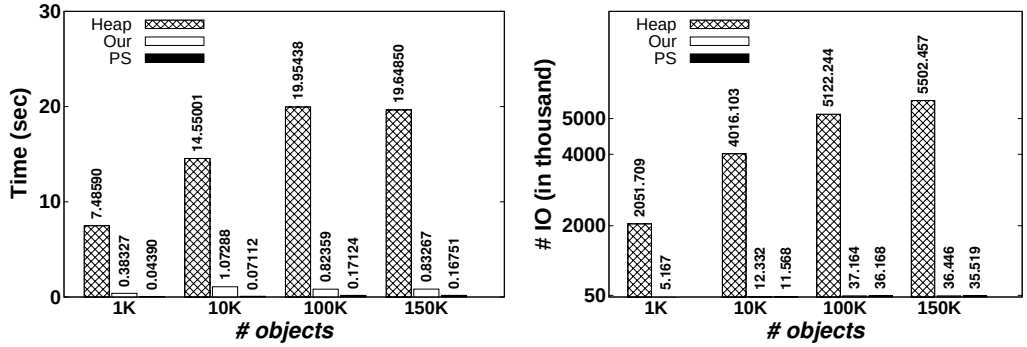
5.4.2 Experiment Result

5.4.2.1 Effect of data cardinality

Fig. 5.8 shows the effect of data cardinality to the performance of the studied algorithms. In Fig. 5.8(a), our algorithm is significantly better than **Heap**. The CPU cost of our algorithm is higher than **PS** algorithm mainly because our algorithm computes the safe zone and the result of the queries, whereas **PS** only computes the query result.

In Fig. 5.8(b), our algorithm incurs slightly more IO than **PS** because our algorithm continues to open the R-tree nodes until all objects are evaluated. In contrast, **PS** stops accessing the R-tree once all query results are retrieved. Nevertheless, both

CPU and IO costs of our algorithm are reasonably close to the CPU and IO costs of **PS**.



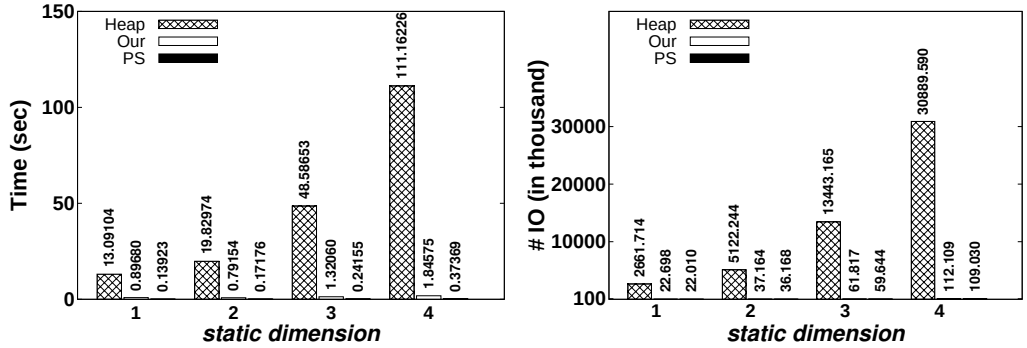
(a) CPU cost (b) IO cost

Figure 5.8: Effect of number of objects

5.4.2.2 Effect of dimensionality

Fig. 5.9(a) shows the CPU cost when the number of static dimensions is changed from 1 to 4. Note that the number of dynamic dimensions, which correspond to the objects' location coordinates are 2. Our algorithm is up to 50 times better than **PS**. Our algorithm also scales better with the increase of the number of static dimensions. The CPU cost of our algorithm is relatively stable for different number of dimensions, because the static score is computed in a single operation regardless of the number of dimensions.

In Fig. 5.9(b), the IO cost of our algorithm is slightly higher than **PS** because it verifies all R-tree nodes to ensure the correctness of the safe zone. However, the IO cost of our algorithm is reasonably close to **PS**, which shows that the safe zone computation does not add large IO overhead.



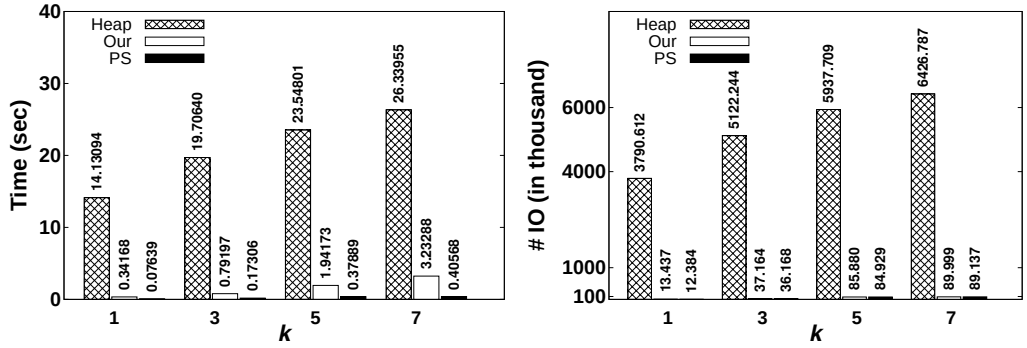
(a) CPU cost (b) IO cost

Figure 5.9: Effect of # static dimension

5.4.2.3 Effect of k

Fig. 5.10 shows the effect of the value of k to the performance of the studied algorithms. As shown in Fig. 5.10(a), the CPU cost of all algorithms increases as the increase of the value of k. This is because all algorithms open more R-tree nodes when k is getting bigger. In addition, our algorithm requires higher CPU cost because for each entry in the result set, it computes the minimum distance between the query and the corresponding preferred regions. The higher value of k, the higher number of entries to be considered.

In Fig. 5.10(b), when k increases, the IO cost of all algorithms increases because they open more R-tree nodes to retrieve the required result size. The IO cost of our algorithm is reasonably close to the IO cost of **PS** regardless of the value of k. It shows that the increase of the IO cost is mainly due to the top-k objects computation.



(a) CPU cost

(b) IO cost

Figure 5.10: Effect of k

5.5 Conclusions

We propose an efficient distance measurement technique that can be applied for complex shapes. We use the technique in our algorithm to construct the safe zone of moving top-k queries. Our algorithm is applicable for any monotonic scoring function. Our extensive experiments on real data set demonstrate that the cost of our proposed algorithm is up to 50 times better than the naïve algorithm.

Chapter 6

Reducing The Communication Cost of Continuous Spatial Queries

This chapter presented algorithms to reduce the communication cost on continuous monitoring of spatial queries. We introduce a *Quiet Zone* for every moving object (client) such that if the object inside the zone, it does not initiate communication with the server, i.e., it remains quiet.

6.1 Overview

With advances in wireless network and the availability of inexpensive mobile devices equipped with position sensing technology, location based services have gained significant interest in the past decades. The rapid development of Internet of Things (IoT) technology allows the expansion of the services due to the growing number of GPS-enabled devices connected to the network. Some applications of location based services include location based advertisement, emergency services, traffic monitoring, army strategic planning, location based games, geo-social networking etc.

In real world scenario, many objects are continuously moving. Therefore, continuous monitoring of spatial queries has received significant research attention recently. Algorithms for continuous monitoring of spatial queries have been proposed in the past few years, such as in range query, nearest neighbour query, reverse nearest neighbour query etc. Most of these studies have focused on reducing the CPU cost of updating the query result in response to the location updates sent by the moving objects.

In this chapter, we study the continuous monitoring of spatial queries over moving

objects, i.e., a scenario where the queries are static whereas the data objects are continuously moving. The moving object could be a user carrying mobile device or, in the context of IoT, a smart machine that is connected to the internet, such as a smart car. Consider the example of a store in an urban city. Suppose the store wants to advertise its special deal, the store may want to continuously monitor customers within 1 Km of its location to inform them about the deal.

Most studies in continuous monitoring of spatial queries use a *client-server* model, where the client sends a query to the server and the server responds to it and sends the query results back to the client. In the traditional client server model, each object updates its location to the server after every t time units (*time stamp*). For example, a restaurant (client) in the urban city may issue a query to the service provider (server) to constantly report all objects within 1 Km range. An object could be a person with mobile device walking down the street or driving around the city. Traditional client server model requires every object to send its location at every time stamp, so that the server has the up to date location of the object and is able to update the query result accordingly. This traditional client-server model creates significant amount of data transmission because every object needs to send its location to the server at every time stamp.

In our solution, we assume a client-server model. However, we do not require the object to update its location at every time stamp. We require an object to send its location only if it affects the query results. In other words, a location update of an object is required if and only if the movement of this object affects the result of at least one query. Otherwise, it will stay quiet and does not initiate a communication with the server.

For each object, we introduce a zone such that as long as the object is inside this zone, it does not affect the result of *any* query in the system. We call such zone as *quiet zone* because the object remains quiet as long as it is inside the zone. Note that computing quiet zone is not trivial because of the following challenges. The first challenge is how to efficiently determine the area where the object does not affect the result of any query, considering that there might be many queries in the system. Secondly, note that the object needs to check if it is inside its quiet zone. Because the object usually has limited resources, for example mobile phones or sensors, therefore the shape of the quiet zone should be simple such that the object can easily check whether it is inside the quiet zone or not. With quiet zone, we significantly reduce the communication cost by at least one order of magnitude.

Most of the previous works on continuous spatial queries have been focused on minimizing the CPU cost. However, note that in highly connected urban city, communication cost is also very high because each object sends its location at every time stamp. We are the first to present a generic framework that focuses on

reducing the communication cost for many different type of spatial queries. Below, we summarize our contributions.

- To the best of our knowledge, we are the first to present a generic framework for continuous spatial queries on client server model that significantly reduces the communication cost.
- We conduct extensive experiment on real data set that shows that our algorithm is one order of magnitude better in terms of communication cost than the traditional approach.

6.2 Proposed framework

To reduce the communication cost, we construct a quiet zone such that if the object inside its quiet zone, it does not affect the query result. At every time stamp, the object checks whether it is inside its quiet zone or not. As long as the object lies inside the quiet zone, it is not required for the object to send its location to the server. The quiet zone should be a simple shape such that the object is able to easily check whether it is inside the zone or not. If the quiet zone is a complex shape, the computation cost at objects may be very high and moving objects such as mobile phones or sensors usually do not have a lot of resources and computation power. Our framework can be used in many different type of spatial queries. For the ease of presentation, we first present our approach for range queries. Subsequently, we show that it can be easily applied for other spatial queries as we briefly describe later.

Given a set of objects O and a query point q with range r, a range query retrieves all objects within r distance from q, i.e., objects inside a circle centered at q with radius r. This circle is called *query circle* and is denoted as c_q (query circle of the query q). The continuous range query reports all objects inside c_q to q at each time stamp.

Consider two moving objects o_1 and o_2 and six range queries q_1, q_2, q_3, q_4, q_5 and q_6 in Fig. 6.1. The query circles c_{q_1} to c_{q_6} are shown shaded. Object o_1 is the query result of q_2 since o_1 is inside c_{q_2}. For the same reason, o_1 is not the query result of q_1, q_3, q_4, q_5 and q_6. Note that o_1 is the query result of q_2 as long as it is inside c_{q_2}. Similarly, o_1 will not be the query result of q_1, q_3, q_4, q_5 and q_6 as long as o_1 is outside c_{q_1}, c_{q_3}, c_{q_4}, c_{q_5} and c_{q_6} respectively. This circumstances applies for o_2 as well. Therefore, the query result of all queries remain unchanged as long as o_1 and o_2 do not enter or leave a query circle. In general, when the movement of an object o does not change the result of all queries, o does not need to update its location to the server. Therefore, the quiet zone of o is bounded by query circles around o.

In Fig. 6.1, the optimal quiet zone of o_1 is the striped area. However, when the

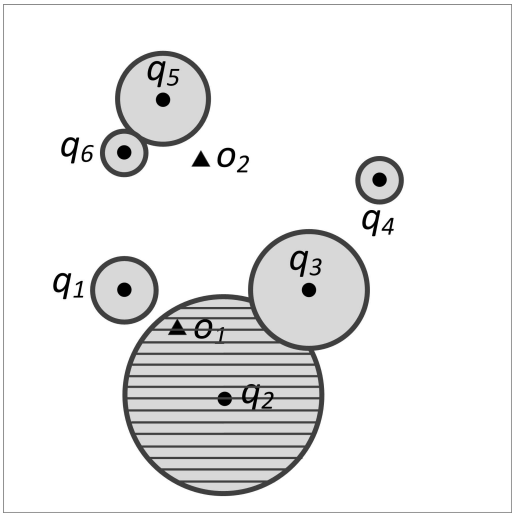


Figure 6.1: Range query

number of pruning circles that contain o_1 is high, the optimal quiet zone becomes complicated since it involves many circles. This, in turn, may cause the computation cost to check if o_1 is inside the quiet zone becomes very high. Similarly, the optimal quiet zone for o_2 is the white area. To check if o_2 is inside its quiet zone, we have to verify o_2 against all query circles in the data space (c_{q_1}, c_{q_3}, c_{q_4}, c_{q_5} and c_{q_6}). This is costly, especially when the number of queries in the system are high. Therefore, we need to approximate the quiet zone such that the computation cost is feasible for the objects.

A simple approach to approximate the quiet zone of o (denoted as Z_o) is to get the closest pruning circle to o and compute a regular shape such as circle, square or rectangle. However, this simple approach results in a small quiet zone for o. Consider object o_1 in Fig. 6.2. The closest pruning circle to o_1 is c_{q_2}. If circle is selected, the quiet zone of o_1 (Z_{o_1}) is the small circle (shown dark shaded) centered at o_1 with radius of the minimum distance between o_1 and c_{q_2}. Similarly for object o_2, if rectangle is selected, the quiet zone of o_2 (Z_{o_2}) is the rectangle (shown dark shaded or dashed line) containing o_2 that is bounded by surrounding query circles.

To improve the size of the quiet zone, we construct a polygon bounded by all query circles around the object. Consider the example in Fig. 6.3(a). The dark shaded area Z_{o_1} is the improved quiet zone of o_1. Similarly, The dark shaded area Z_{o_2} in Fig 6.3(b) is the improved quiet zone of o_2. Steps to compute the improved quiet zone are presented as follows.

For each query circle c_q that does not contain o, a *tangent line* is computed. Given an object o and a query point q with corresponding query circle c_q, the tangent line on c_q with respect to o (denoted as $l_{o:q}$) is the tangent line of c_q that is perpendicular to the line connecting o and q. Consider object o_1 and c_{q_3} in Fig. 6.3(a). The line

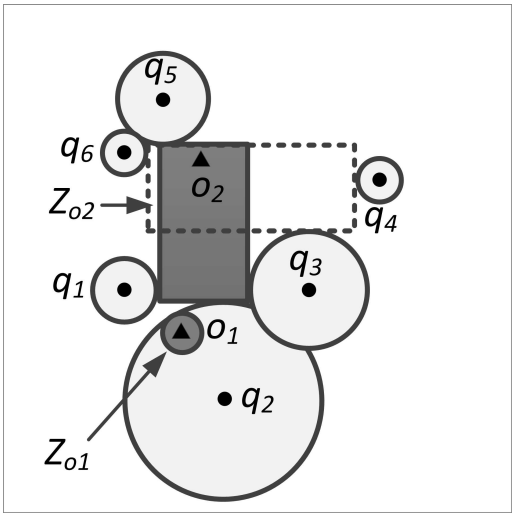
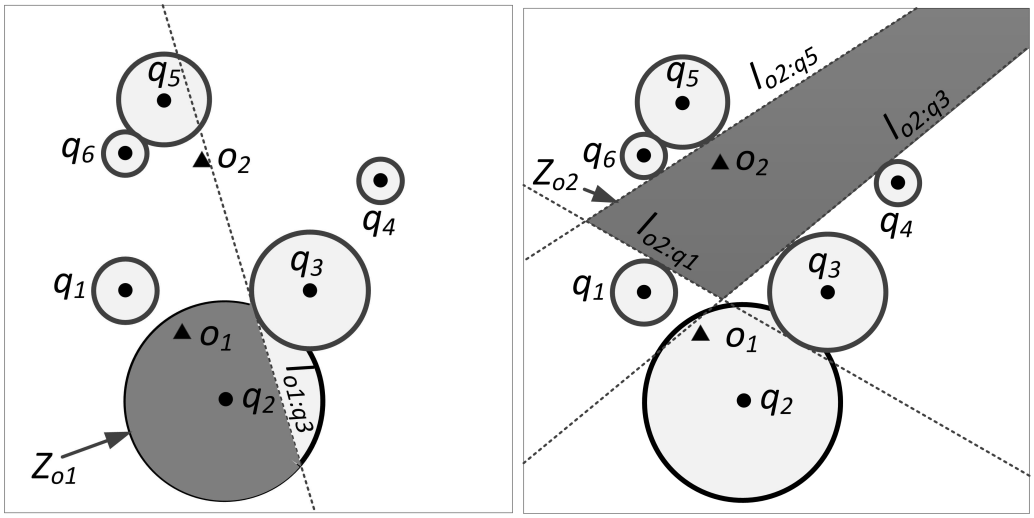


Figure 6.2: Basic quiet zone

$l_{o_1:q_3}$ shown in broken is the tangent line of c_{q_3} with respect to o_1. Similarly in Fig. 6.3(b), $l_{o_2:q_1}$, $l_{o_2:q_3}$ and $l_{o_2:q_4}$ are the tangent lines of c_{q_2}, c_{q_3} and c_{q_4} with respect to o_2. The tangent lines $l_{o_2:q_4}$ and $l_{o_2:q_6}$ are not shown because they are coincide with $l_{o_2:q_3}$ and $l_{o_2:q_5}$ respectively.

On the other side, if a query circle contains o, we store it in a set called *circle set* of o. The quiet zone of o is the intersection area of the followings: (i) a polygon that does not overlap all query circles which do not contain o, and (ii) a set of query circles that contain o.



(a) Quiet zone of o_1 (b) Quiet zone of o_2

Figure 6.3: Improved quiet zone

6.3 Computing Quiet Zone

One straight forward approach to compute the quiet zone is to consider each query in the system. To get the quiet zone of an object o, we draw a tangent line on all query circles with respect to o. The quiet zone of o is a polygon containing o constructed by all tangent lines as shown in Fig 6.4.

However, this straight forward approach is too expensive. It requires computing the tangent line of all query circles in the system. Note that there are queries that do not contribute to the quiet zone. In Fig 6.4, the tangent line $l_{o:q_2}$ does not overlap the quiet zone of o (shown as shaded). After q_1, q_4, q_5 and q_8 are processed, $l_{o:q_2}$ will not affect the current quiet zone. In other words, if q_2 is ignored, the quiet zone of o can still be computed correctly.

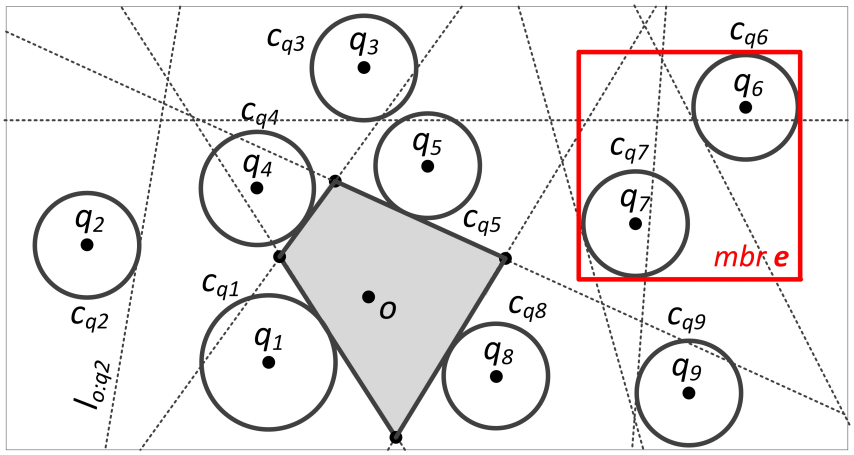


Figure 6.4: Computing quiet zone

Next, we present an observation to help us constructing the quiet zone efficiently. Without loss of generality, we assume that the data space is bounded by a square. Query circles are indexed in R-tree (denoted as Q-tree). Q-tree is stored in the memory and is updated when a query is issued or is deleted from the server.

Observation 1. Given an object o, the quiet zone of o (Z_o), a query point q with corresponding query circle c_q, q can be ignored if c_q does not overlap Z_o. Similarly, an entry e in Q-tree can be ignored if e does not overlap Z_o

The proof is obvious and is omitted from this chapter. Consider q_2 in Fig 6.4. Since c_{q_2} does not overlap the quiet zone of o (shown shaded), q_2 is ignored. Hence, $l_{o:q_2}$ is not necessarily computed and the quiet zone of o remains unchanged. Similarly for an entry e (red rectangle) of the Q-tree. Since e does not overlap the quiet zone, it is ignored and its children are not evaluated.

6.4 Extension for Other Spatial Queries

A similar study to efficiently monitor the nearest neighbour of queries has been presented by Mouratidis *et al.* [156]. Their solution can be easily integrated with our proposed solution. In this section, we show that our approach can be easily extended for other location-based queries. Specifically, we present the extension of our approach to continuously monitor the result of reverse k nearest neighbour query (RkNN), window query and reverse approximate nearest neighbour query [12].

To illustrate how to extend our proposed algorithm for RkNN query, we use *influence zone (infZone)*, the state of the art algorithm for continuous monitoring of RkNN. First, we compute a smallest circle that contains all vertices of the influence zone. This circle is called *smallest enclosing circle* (sec) [170]. Consider q_1 and q_2 in Fig. 6.5. $infZone_{q_1}$ and $infZone_{q_2}$ are the influence zone of q_1 and q_2 respectively (shown shaded). Circles sec_{q_1} and sec_{q_2} are the smallest enclosing circle of $infZone_{q_1}$ and $infZone_{q_2}$. Once all smallest enclosing circles are computed, our proposed algorithm can be used to construct the quiet zone of objects. Similarly for window query, the smallest enclosing circle that contains the query rectangle is computed and used to construct the quiet zone. In addition, our proposed algorithm can be directly applied for reverse approximate nearest neighbour (RANN) query. This is because RANN query computes circle-shaped pruning regions that can be directly used to construct the quiet zone of objects.

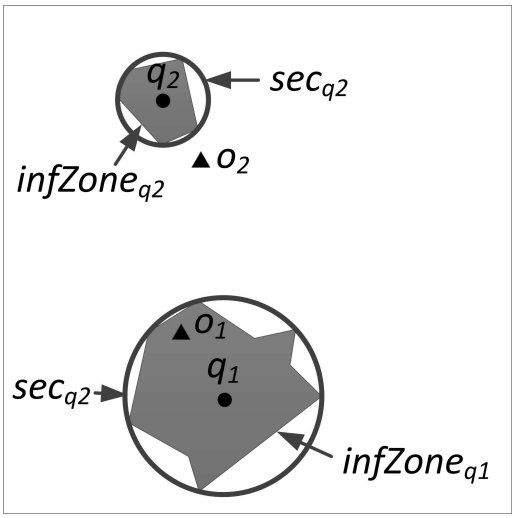


Figure 6.5: Extension for RkNN query

6.5 Algorithm

In this section, we present our algorithm to efficiently compute the quiet zone of an object. Algorithm 8 provides the details. Initially, the quiet zone of an object o (Zo) is set to cover the whole data space.

The root of Q-tree is inserted to a heap sorted according to the minimum distance from o (line 2). The entries in Q-tree are iteratively accessed from the heap. If the de-heaped entry e is an intermediate node or a leaf node and it completely lies outside Vo, it is ignored. Otherwise, its children are inserted to the heap (line 7). If the entry is a query circle and it overlaps Vo, it is used to update Zo (line 10). The algorithm stops when the heap becomes empty.

Algorithm 8 Compute quiet zone

```
 1: Zo = Vo = {Vertices of the data space}
 2: insert root of Q-tree in a min-heap h
 3: while h is not empty do
 4:    de-heap an entry e
 5:    if e is an intermediate or leaf node then
 6:       if e overlaps Vo then
 7:          insert all children of e in h
 8:    else                                          ▷ e is a query circle
 9:       if c_e overlaps Vo then
10:          update Zo
11: return Zo
```

Now, we present Algorithm 9 to describe how to update the quiet zone of o when a query q is considered, i.e., c_q overlaps Zo. First, we explain the case when c_q contains the object. If o is inside c_q, the query is recorded in the query set of object (line 2). For each object o, we maintain a set So to store the queries whose query circle contains o, i.e., o is one of the query results. In this case, the polygon Vo remains unchanged.

Next, we describe the case when c_q does not contain o. Initially, the tangent line of c_q with respect to o ($l_{o:q}$) is computed (line 4). For each edge e, we compute a line (l) using two end points of e. Then, the intersection point (p) between l and ($l_{o:q}$ is computed. If p lies on e, it is inserted into Vo (line 7). Then, for each vertex in Vo, if the vertex lies outside updated Vo, it is removed from Vo (line 10). Algorithm 9 returns the updated quiet zone of o. The final quiet zone of o is the intersection region between Vo and So.

Example 1. Figures 6.6-6.9 show the construction of the quiet zone of object o. In Fig. 6.6, the quiet zone is initially set to be the whole data space bounded by four vertices v_1 to v_4 ($Zo = Vo = \{v_1 \cdots v_4\}$) . When the nearest query is encountered (q_1), the algorithm checks if c_q overlaps Zo. Since c_{q_1} overlaps Zo and o is outside c_{q_1},

Algorithm 9 Update quiet zone

```
1: if c_q contains o then
2:    insert o in So
3: else
4:    compute the tangent line l_{o:q}
5:    for each edge e do
6:       p ← intersection point of e and l_{o:q}
7:       insert p to Vo
8:    for each v_i ∈ V_o do
9:       if v_i is outside Vo then
10:         remove v_i from Vo
11: return Zo = V_o ∩ S_o
```

a tangent line of c_{q_1} with respect to o ($l_{o:q_1}$) is computed. In Fig. 6.7, $l_{o:q_1}$ updates the polygon Vo and the new quiet zone contains vertices v_1, v_2, v_3, v_5 and v_6. At this stage, the quiet zone is $Zo = Vo = \{v_1, v_2, v_3, v_5, v_6\}$ as shown shaded in Fig. 6.7. The algorithm continues with q_2. Since c_{q_2} overlaps Zo and o is inside c_{q_2}, q_2 is inserted in the query set of o (S_o). Zo is then updated to be the intersection of Vo and c_{q_2} (shown dark shaded in Fig. 6.8). Next, the algorithm checks q_3. Similar to q_1, $l_{o:q_3}$ is computed and is used to update Vo. The polygon Vo now contains vertices $\{v_1, v_7, v_8, v_6\}$. The algorithm continues to check q_4. Since c_{q_4} does not overlap Zo, it is ignored. Since all queries have been considered, the algorithm terminates. The final quite zone is $Zo = V_o \cap S_o$ (shown dark shaded in Fig. 6.9).

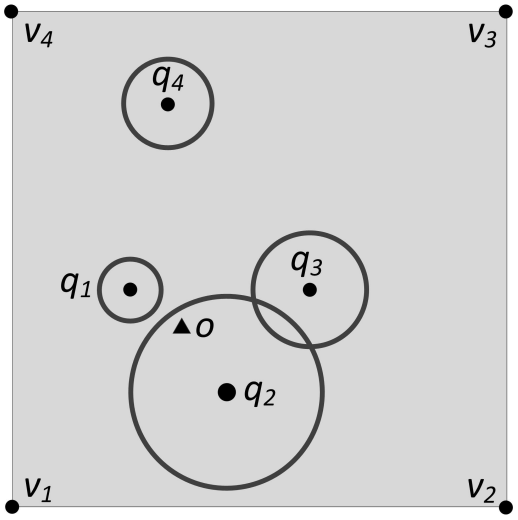


Figure 6.6: Initial zone

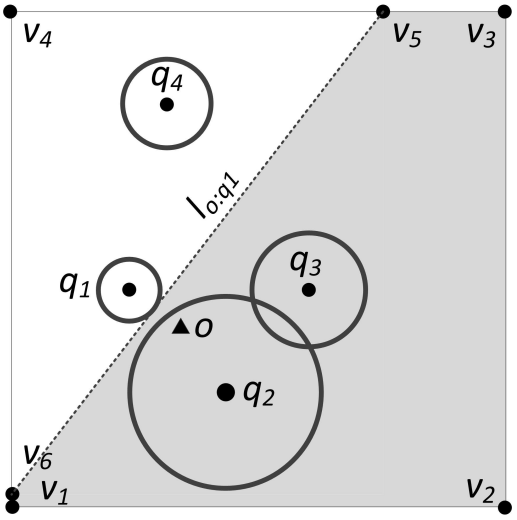


Figure 6.7: Processing q_1

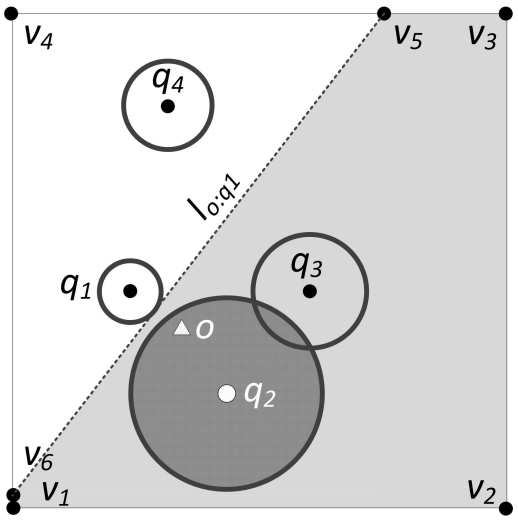


Figure 6.8: Processing q_2

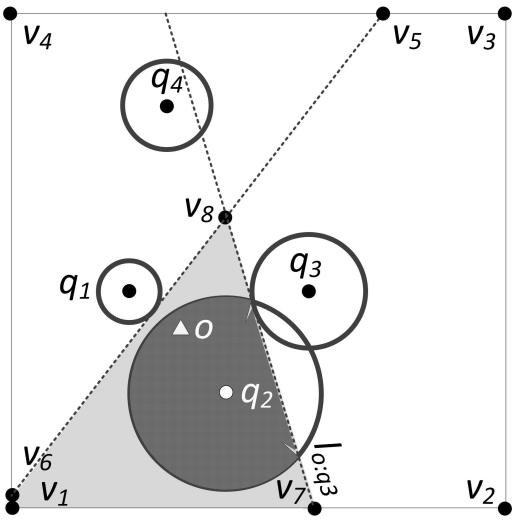


Figure 6.9: Final quiet zone

6.6 Experimental Study

6.6.1 Experimental Setting

The main concern of our algorithm (**QZone**) is to reduce the total communication between moving objects and the server. We compare **QZone** with the traditional time stamp approach (**Traditional**). In **Traditional**, each object reports its position to the server at every time stamp.

All algorithms are implemented in C++. The experiments are run on a 64-bit machine with Intel Core *I5* 2.3GHz and 8GB memory running on Debian Linux. We use a real data set containing 100,000 points of interest (facility) from Los Angeles (LA) [171]. The moving objects are generated by simulating moving cars in the road network of LA using Brinkhoff data generator [164]. We randomly select 500 facilities and treat them as the query points.

We report the communication cost incurred in both algorithms during 500 time stamp observation. We assume that the length of time stamp is one second. In **QZone**, when an object updates its location, i.e., sends its new location to the server, the server computes the new quiet zone and sends it to the object. Hence, the communication cost in **QZone** is multiplied by two. We also report the average CPU cost required by an object to check if it is inside its quiet zone. From our experiment, we show that the checking cost is very small and is feasible for the mobile devices that limited in resources.

We study the performance of both algorithm in different experiment settings. The parameters used in the experiments are shown in Table 6.1 and the default values are shown in bold.

Table 6.1: Experiment Parameters

Parameters	**Range**
Range (Km)	$0.1, 0.5, \mathbf{1}, 5, 10$
Number of facilities (X 1000)	$1, \mathbf{10}, 50, 100$
Number of users (X 1000)	$1, 10, \mathbf{50}, 100$
Users' speed (Km/hr)	$40, 60, \mathbf{80}, 100, 120$
Number of Queries	$100, \mathbf{500}, 1000, 2000$

6.6.2 Experiment Result

6.6.2.1 Effect of query size

In Fig. 6.10, we show the performance of our algorithm in different query sizes. **QZone** significantly reduces the communication cost up to 200 times. The communication cost slightly increases as the increase of number of queries. The reason is that the increase of query size makes the size of quiet zone becomes smaller. In **Traditional**, the communication cost remains the same because each object updates its location to the server at every time stamp.

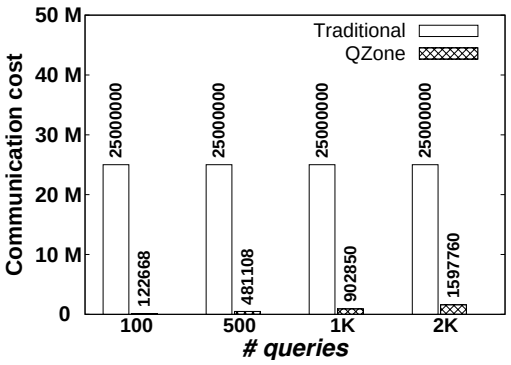
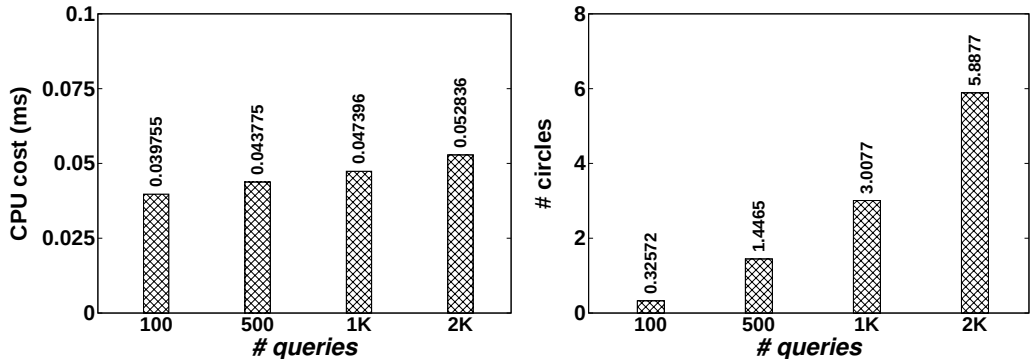


Figure 6.10: Effect of # queries

Fig. 6.11 shows the effectiveness of quiet zone. In Fig. 6.11(a), we show that the CPU cost that is required to check if an object is inside the quiet zone is very small, around 0.05 ms. It indicates that our algorithm is feasible for mobile devices which generally have limited resources. In Fig. 6.11(b), we show that the average size of circle set is quiet small (around 5 for 2000 queries). As expected, the number of query circles that contain the object increases as the increase of the number of queries. The increase of the circle set size also contributes to the decrease of the size of quiet zone.

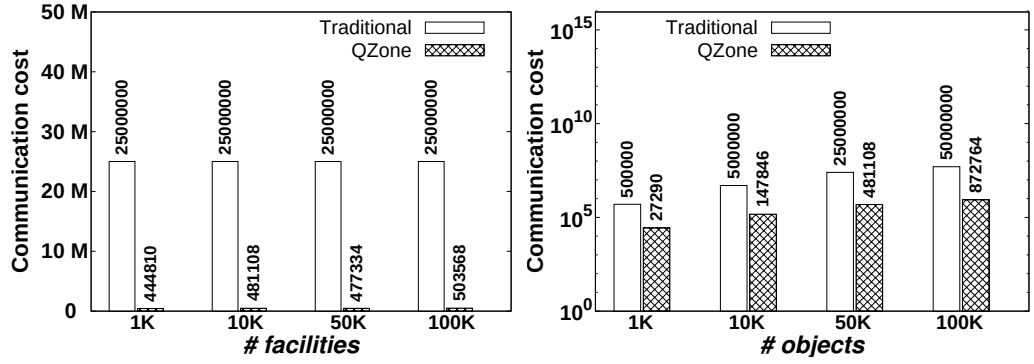


(a) Checking cost (b) Average # circles

Figure 6.11: Effectiveness of quiet zone

6.6.2.2 Effect of data size

In Fig. 6.12(a), we compare the communication cost of **QZone** with the cost of **Traditional** for different number of facilities. Our algorithm is up to 50 times better than **Traditional**. The cost generally increases when the number of facility increases. This is because the increase of number of facilities makes the density of the query increases.



(a) # facilities (b) # moving objects

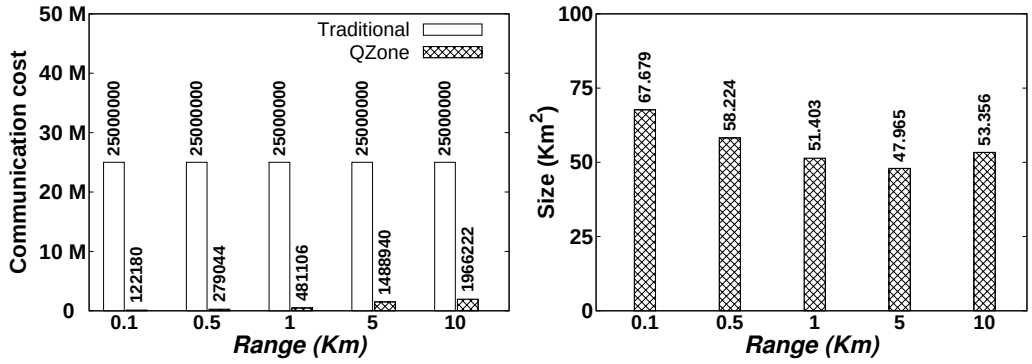
Figure 6.12: Effect of data size

Fig. 6.12(b) shows the communication cost on both algorithms for different number of moving objects. Note that the figure is shown in log scale. **QZone** is at least one order of magnitude better than **Traditional**. The cost difference between two algorithms increases as the increase of number of objects. **QZone** scales significantly better than **Traditional**.

6.6.2.3 Effect of range

Fig. 6.13(a) shows the communication cost of **QZone** and **Traditional** for different query ranges. **QZone** is up to 200 times better than **Traditional**. In **QZone**, the

cost increases as the increase of the range. One reason is that the increase of query range makes the size of query circles increases. This in turn causes the size of quiet zone decreases as shown in Fig. 6.13(b). Fig. 6.13(b) also shows that when the query range is set to be significantly large (10 Km), the size of the quiet zone becomes larger because the average size of query circles that contain the object becomes significantly larger.



(a) Communication cost (b) QZone size

Figure 6.13: Effect of query range

6.6.2.4 Effect of speed

Fig. 6.14 shows the communication cost of both algorithms for different object's speed. **QZone** is up to 90 times better than **Traditional**. The communication cost increases as the increase of object's speed. This is because faster object requires less time to leave its quiet zone than the slower object. It makes the object with faster speed sends location updates to the server more frequent than the slower one.

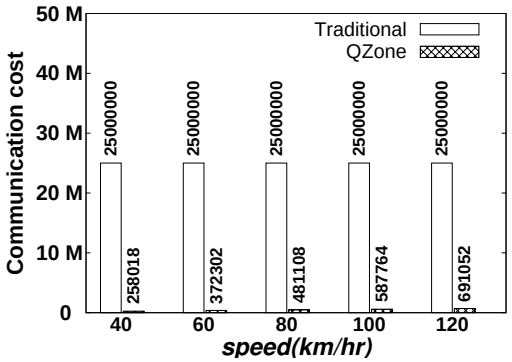


Figure 6.14: Effect of speed

6.7 Conclusion

In this chapter, we propose a generic framework to reduce the communication cost of many types of spatial queries. We propose technique and algorithm to efficiently compute the quiet zone of the moving object. We demonstrate that the cost to check if object is inside the quiet zone is reasonably low. The experimental results showed that our algorithm is significantly better in terms of communication cost than the traditional approach.

Chapter 7

Concluding Remarks

7.1 Conclusion

In this thesis, we study two types of query: queries to find the most influential facilities for a user, and queries to find the influenced users given a query facility. We also study the continuous monitoring of spatial queries that focus on reducing the computation and communication costs. We give the details of our contributions below.

In Chapter 3, we propose a new definition of influence where a user u is said to be influenced by not only its closest facility but also every other facility that is almost as close to u as its closest facility is. We introduce reverse approximate nearest neighbor (RANN) query to compute the influence set of a facility by considering relative distance between users and facilities. We show that the existing pruning techniques cannot be applied for RANN problem. We therefore propose efficient and tight pruning techniques and use them in our RANN algorithm. Our experimental study on real and synthetic data sets shows that our algorithm is several orders of magnitude better than the naïve algorithm as well as the significantly improved version of naïve algorithm.

In Chapter 4, we extend our work on snapshot RANN queries for continuous RANN queries. We propose an efficient monitoring technique that utilizes the Voronoi diagram of facilities. Our technique checks only one facility to verify if a user is an RANN of a query. We extend the RkNN's state-of-the-art algorithm for continuous RANN queries and compare its performance with our algorithm. Our extensive experiment result shows that our Voronoi-based algorithm significantly outperforms the competitor.

In Chapter 5, we present a safe zone based approach to efficiently monitor the k most influential facilities for a user when distance and some other factors are taken into consideration. We propose a technique to compute the minimum

distance between a point and an arbitrary shape. We use the technique to design a generic algorithm that works for any monotonic scoring function. The result of our experiment demonstrates that the cost of our algorithm is up to 50 times better than the naïve algorithm.

In Chapter 6, we present a generic framework to reduce the communication cost of many different variety of continuous spatial queries, such as range query, window query, nearest neighbor query and reverse approximate nearest neighbor query. We show that checking cost at object is reasonably low so that our framework is feasible for moving devices which commonly have limited resources. Our experimental study shows that our algorithm significantly reduces the communication cost.

7.2 Future Work

Below, we propose several possible directions for future work.

7.2.1 Reverse Approximate Top-k (RATk) Query

In Chapter 3, we present RANN queries to compute the influence set of a facility when distance is the only considered factors. In many real world scenario, users may consider other factors in choosing a facility, such as price, rating etc. Hence, it will be interesting to study the reverse approximate top-k queries where the objective is to find every user for which the query facility is an approximate top-k facility. In other words, the RATk queries return every user for which the score of the query facility is almost as good as her most preferred facility.

7.2.2 RANN and RATk Queries in Road Network

In Chapter 3, we propose a new definition of influence using relative distance between the users and the facilities. We introduce RANN query as another perspective to capture the notion of influence. We propose our technique and algorithm to solve RANN query in Euclidean distance. A possible future work is to study the RANN queries in road network. The problem definition mentioned in Section 3.2 is relevant in road network setting. However, our proposed techniques and algorithms in Chapter 3 are not applicable for road network distance. Similarly, RATk in road networks is also an interesting topic for further study.

7.2.3 Improving the Effectiveness of Quiet Zone

In Chapter 6, we propose a quiet zone which is an area such that an object does not need to send its location to the server as long as it is inside its quiet zone. As mentioned in Section 6.5, to get the quiet zone, we compute a tangent line using

the closest point on the query circles from the object. The size of the quiet zone can be further improved by computing more tangent points using some points on the query circles. Given an object o and a query circle c, we need to compute the appropriate number and position of tangent lines such that the quiet zone is maximized. Computing more tangent points will increase the cost, however it may significantly increase the size of the quiet zone. Therefore, it will be interesting to analyze the trade-off between computation cost and communication cost.

Bibliography

[1] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pages 802–813, 2003.

[2] Muhammad Aamir Cheema. Circulartrip and arctrip: Effective grid access methods for continuous spatial queries. Master's thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney Australia, 3 2007.

[3] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, and Wei Wang. Reverse k nearest neighbors query processing: Experiments and analysis. *PVLDB*, 2015.

[4] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse knn search in arbitrary dimensionality. *PVLDB*, pages 744–755, 2004.

[5] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, pages 577–588, 2011.

[6] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, and Ying Zhang. SLICE: Reviving regions-based pruning for reverse k nearest neighbors queries. In *ICDE*, pages 760–771, 2014.

[7] Ralf Hartmut Güting. An introduction to spatial database systems. *VLDB J.*, 3(4):357–399, 1994.

[8] Rone Ilídio da Silva, Daniel Fernandes Macedo, and José Marcos S. Nogueira. Spatial query processing in wireless sensor networks - A survey. *Information Fusion*, 15:32–43, 2014.

[9] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1):364–372, 2010.

[10] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[11] Arif Hidayat, Muhammad Aamir Cheema, and David Taniar. Relaxed reverse nearest neighbors queries. In *SSTD*, 2015.

[12] Arif Hidayat, Shiyu Yang, Muhammad Aamir Cheema, and David Taniar. Reverse approximate nearest neighbor queries. *IEEE Transactions on Knowledge and Data Engineering*, PP(99):1–1, 2017.

[13] Arif Hidayat and Muhammad Aamir Cheema. Quiet zone: Reducing the communication cost of continuous spatial queries. In *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics, UrbanGIS@SIGSPATIAL 2017, Redondo Beach, CA, USA, November 7, 2017*, 2017.

[14] Dongsheng Li, Jiannong Cao, Xicheng Lu, and Kaixian Chen. Efficient range query processing in peer-to-peer systems. *IEEE Trans. Knowl. Data Eng.*, 21(1):78–91, 2009.

[15] Apostolos Papadopoulos and Yannis Manolopoulos. Multiple range query optimization in spatial databases. In *Advances in Databases and Information Systems, Second East European Symposium, ADBIS'98, Poznan, Poland, Spetember 7-10, 1998, Proceedings*, pages 71–82, 1998.

[16] Jing Shan, Donghui Zhang, and Betty Salzberg. On spatial-range closest-pair query. In *Advances in Spatial and Temporal Databases, 8th International Symposium, SSTD 2003, Santorini Island, Greece, July 24-27, 2003, Proceedings*, pages 252–269, 2003.

[17] Hoong Kee Ng and Hon Wai Leong. Path-based range query processing using sorted path and rectangle intersection approach. In *Database Systems for Advances Applications, 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2004, Proceedings*, pages 184–189, 2004.

[18] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Continuous monitoring of distance-based range queries. *IEEE Trans. Knowl. Data Eng.*, 23(8):1182–1199, 2011.

[19] Dragan Stojanovic, Apostolos N. Papadopoulos, Bratislav Predic, Slobodanka Djordjevic-Kajan, and Alexandros Nanopoulos. Continuous range monitoring of mobile objects in road networks. *Data Knowl. Eng.*, 64(1):77–100, 2008.

[20] Fuyu Liu, Tai T. Do, and Kien A. Hua. Dynamic range query in spatial network environments. In *DEXA*, pages 254–265, 2006.

[21] Haojun Wang and Roger Zimmermann. Snapshot location-based query processing on moving objects in road networks. In *GIS*, page 50, 2008.

[22] Hans-Peter Kriegel, Peer Kröger, and Matthias Renz. Continuous proximity monitoring in road networks. In *GIS*, page 12, 2008.

[23] Axel Küpper and Georg Treu. Efficient proximity and separation detection among mobile targets for supporting location-based community services. *Mobile Computing and Communications Review*, 10(3):1–12, 2006.

[24] Bugra Gedik and Ling Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *EDBT*, pages 67–87, 2004.

[25] Ying Cai, Kien A. Hua, and Guohong Cao. Processing range-monitoring queries on heterogeneous mobile objects. In *Mobile Data Management*, 2004.

[26] Xiaoyuan Wang and Wei Wang. Continuous expansion: Efficient processing of continuous range monitoring in mobile environments. In *DASFAA*, pages 890–899, 2006.

[27] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku. Distributed continuous range query processing on moving objects. In *DEXA*, pages 655–665, 2006.

[28] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, 1984.

[29] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 71–79, 1995.

[30] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[31] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 154–165, 1998.

[32] Christian S. Jensen, Jan Kolárvr, Torben Bach Pedersen, and Igor Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

[33] M. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[34] Mohammad R. Kolahdouzan and Cyrus Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *Spatio-Temporal Database Management, 2nd International Workshop STDBM'04, Toronto, Canada, August 30, 2004*, pages 33–40, 2004.

[35] Cyrus Shahabi, Mohammad R. Kolahdouzan, and Mehdi Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. In *ACM-GIS*, pages 94–10, 2002.

[36] Hyung-Ju Cho and Chin-Wan Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, pages 865–876, 2005.

[37] Haibo Hu, Dik Lun Lee, and Jianliang Xu. Fast nearest neighbor search on road networks. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 186–203, 2006.

[38] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.

[39] Hua Lu, Xin Cao, and Christian S. Jensen. A foundation for efficient indoor distance-aware query processing. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 438–449, 2012.

[40] Xike Xie, Hua Lu, and Torben Bach Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 434–445, 2013.

[41] Jiao Yu, Wei-Shinn Ku, Min-Te Sun, and Hua Lu. An RFID and particle filter-based indoor spatial query evaluation system. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 263–274, 2013.

[42] Joon-Seok Kim and Ki-Joune Li. Location k-anonymity in indoor spaces. *GeoInformatica*, 20(3):415–451, 2016.

[43] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Query processing in spatial databases containing obstacles. *International Journal of Geographical Information Science*, 19(10):1091–1111, 2005.

[44] Chenyi Xia, David Hsu, and Anthony K. H. Tung. A fast filter for obstructed nearest neighbor queries. In *Key Technologies for Data Management, 21st British National Conference on Databases, BNCOD 21, Edinburgh, UK, July 7-9, 2004, Proceedings*, pages 203–215, 2004.

[45] Sarana Nutanong, Egemen Tanin, and Rui Zhang. Visible nearest neighbor queries. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings*, pages 876–883, 2007.

[46] Yunjun Gao, Baihua Zheng, Wang-Chien Lee, and Gencai Chen. Continuous visible nearest neighbor queries. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 144–155, 2009.

[47] Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, pages 337–348, 2007.

[48] Yuan-Ko Huang, Shi-Jei Liao, and Chiang Lee. Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction. In *SSDBM*, pages 549–557, 2008.

[49] Yuan-Ko Huang, Shi-Jei Liao, and Chiang Lee. Evaluating continuous k-nearest neighbor query on moving objects with uncertainty. *Inf. Syst.*, 34(4-5):415–437, 2009.

[50] Goce Trajcevski, Roberto Tamassia, Hui Ding, Peter Scheuermann, and Isabel F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 874–885, 2009.

[51] Cyrus Shahabi, Lu An Tang, and Songhua Xing. Indexing land surface for efficient knn query. *PVLDB*, 1(1):1020–1031, 2008.

[52] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Kai Xu, and Xuemin Lin. Surface k-nn query processing. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 78, 2006.

[53] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Qing Liu, Kai Xu, and Xuemin Lin. A multi-resolution surface distance model for k-nn query processing. *VLDB J.*, 17(5):1101–1119, 2008.

[54] Songhua Xing, Cyrus Shahabi, and Bei Pan. Continuous monitoring of nearest neighbors on land surface. *PVLDB*, 2(1):1114–1125, 2009.

[55] Andreas Henrich. A distance scan algorithm for spatial access structures. In *ACM-GIS*, pages 136–143, 1994.

[56] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot L. Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 215–226, 1996.

[57] Apostolos Papadopoulos and Yannis Manolopoulos. Performance of nearest neighbor queries in r-trees. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, pages 394–408, 1997.

[58] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 369–380, 1997.

[59] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 43–54, 2008.

[60] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 397–410, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[61] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K.*, pages 422–432, 1997.

[62] George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. Nearest neighbor queries in a mobile environment. In *Spatio-Temporal Database Management, International Workshop STDBM'99, Edinburgh, Scotland, September 10-11, 1999, Proceedings*, pages 119–134, 1999.

[63] Zhexuan Song and Nick Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD*, pages 79–96, 2001.

[64] Xiaopeng Xiong, Mohamed F. Mokbel, Walid G. Aref, Susanne E. Hambrusch, and Sunil Prabhakar. Scalable spatio-temporal continuous query processing for location-aware services. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece*, pages 317–326, 2004.

[65] Yufei Tao and Dimitris Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD Conference*, pages 334–345, 2002.

[66] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. López. Indexing the positions of continuously moving objects. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 331–342, 2000.

[67] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pages 790–801, 2003.

[68] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *VLDB J.*, 15(3):229–249, 2006.

[69] Katerina Raptopoulou, Apostolos Papadopoulos, and Yannis Manolopoulos. Fast nearest-neighbor query processing in moving-object databases. *GeoInformatica*, 7(2):113–137, 2003.

[70] Glenn Simmons Iwerks, Hanan Samet, and Kenneth P. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pages 512–523, 2003.

[71] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and Bin Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 611–622, 2004.

[72] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[73] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining ranked inputs in practice. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 950–961, 2002.

[74] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 281–290, 2001.

[75] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.

[76] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.*, 13(1):49–70, 2004.

[77] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 61–72, 2006.

[78] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 648–659, 2004.

[79] Giuseppe Amato, Fausto Rabitti, Pasquale Savino, and Pavel Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, 21(2):192–227, 2003.

[80] Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 886–895, 2007.

[81] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 896–905, 2007.

[82] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[83] Zhen Zhang, Seung-won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A. Lang, and Yuan-Chi Chang. Boolean + ranking: querying a database by k-constrained optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 359–370, 2006.

[84] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 656–665, 2008.

[85] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[86] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. Joint top-k spatial keyword query processing. *IEEE Trans. Knowl. Data Eng.*, 24(10):1889–1903, 2012.

[87] Dingming Wu, Gao Cong, and Christian S. Jensen. A framework for efficient spatial web object retrieval. *VLDB J.*, 21(6):797–822, 2012.

[88] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 155–162, 2005.

[89] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 901–912, 2013.

[90] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top K spatial keyword search. *IEEE Trans. Knowl. Data Eng.*, 28(7):1706–1721, 2016.

[91] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases - 12th International Symposium, SSTD 2011, Minneapolis, MN, USA, August 24-26, 2011, Proceedings*, pages 205–222, 2011.

[92] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. Scalable top-k spatial keyword search. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 359–370, 2013.

[93] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[94] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 301–310, 2001.

[95] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 275–286, 2002.

[96] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[97] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. Approaching the skyline in Z order. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 279–290, 2007.

[98] Ke Deng, Xiaofang Zhou, and Heng Tao Shen. Multi-source skyline query processing in road networks. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 796–805, 2007.

[99] Lei Zou, Lei Chen, M. Tamer Özsu, and Dongyan Zhao. Dynamic skyline queries in large graphs. In *Database Systems for Advanced Applications, 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part II*, pages 62–78, 2010.

[100] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 261–272, 2010.

[101] Xuegang Huang and Christian S. Jensen. In-route skyline querying for location-based services. In *Web and Wireless Geographical Information Systems, 4th InternationalWorkshop, W2GIS 2004, Goyang, Korea, November 2004, Revised Selected Papers*, pages 120–135, 2004.

[102] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003.

[103] Zhiyong Huang, Hua Lu, Beng Chin Ooi, and Anthony K. H. Tung. Continuous skyline queries for moving objects. *IEEE Trans. Knowl. Data Eng.*, 18(12):1645–1658, 2006.

[104] Yu-Ling Hsueh, Roger Zimmermann, and Wei-Shinn Ku. Efficient updates for continuous skyline computations. In *Database and Expert Systems Applications, 19th International Conference, DEXA 2008, Turin, Italy, September 1-5, 2008. Proceedings*, pages 419–433, 2008.

[105] Mu-Woong Lee and Seung-won Hwang. Continuous skylining on volatile moving data. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1568–1575, 2009.

[106] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. A safe zone based approach for monitoring moving skyline queries. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 275–286, 2013.

[107] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[108] Mikhail J. Atallah and Yinian Qi. Computing all skyline probabilities for uncertain data. In *PODS*, pages 279–287, 2009.

[109] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, and Jeffrey Xu Yu. Probabilistic skyline operator over sliding windows. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1060–1071, 2009.

[110] Wenjie Zhang, Aiping Li, Muhammad Aamir Cheema, Ying Zhang, and Lijun Chang. Probabilistic n-of-n skyline computation over uncertain data streams. In *Web Information Systems Engineering - WISE 2013 - 14th International Conference, Nanjing, China, October 13-15, 2013, Proceedings, Part II*, pages 439–457, 2013.

[111] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.

[112] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop*, 2000.

[113] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[114] Wei Wu, Fei Yang, Chee Yong Chan, and Kian-Lee Tan. Continuous reverse k-nearest-neighbor monitoring. In *MDM*, pages 132–139, 2008.

[115] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *IDEAS*, pages 44–53, 2002.

[116] Tian Xia and Donghui Zhang. Continuous reverse nearest neighbor monitoring. In *ICDE*, pages 77–86, 2006.

[117] James M. Kang, Mohamed F. Mokbel, Shashi Shekhar, Tian Xia, and Donghui Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007.

[118] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, and Wenjie Zhang. Lazy updates: An efficient technique to continuously monitoring reverse knn. *PVLDB*, pages 1138–1149, 2009.

[119] Muhammad Aamir Cheema, Xuemin Lin, Wei Wang, Wenjie Zhang, and Jian Pei. Probabilistic reverse nearest neighbor queries on uncertain data. *IEEE Trans. Knowl. Data Eng.*, 2010.

[120] Xiang Lian and Lei Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *VLDB J.*, 18(3):787–808, 2009.

[121] Thomas Bernecker, Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz, and Andreas Züfle. A novel probabilistic pruning approach to speed up similarity queries in uncertain databases. In *ICDE*, pages 339–350, 2011.

[122] Thomas Bernecker, Tobias Emrich, Hans-Peter Kriegel, Matthias Renz, , and Stefan Zankl Andreas Züfle. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. *PVLDB*, pages 669–680, 2011.

[123] Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis, and Yufei Tao. Reverse nearest neighbors in large graphs. *IEEE Trans. Knowl. Data Eng.*, pages 540–553, 2006.

[124] Huan-Liang Sun, Chao Jiang, Jun-Ling Liu, and Limei Sun. Continuous reverse nearest neighbor queries on moving objects in road networks. In *WAIM*, pages 238–245, 2008.

[125] Guohui Li, Yanhong Li, Jianjun Li, LihChyun Shu, and Fumin Yang. Continuous reverse k nearest neighbor monitoring on moving objects in road networks. *Inf. Syst.*, 35(8):860–883, 2010.

[126] Arif Hidayat, Muhammad Aamir Cheema, and David Taniar. Relaxed reverse nearest neighbors queries. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings*, pages 61–79, 2015.

[127] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.

[128] Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Monitoring reverse top-k queries over mobile devices. In *Proceedings of the Tenth ACM International Workshop on Data Engineering for Wireless and Mobile Access, MobiDE 2011, Athens, Greece, June 12, 2011*, pages 17–24, 2011.

[129] Shen Ge, Leong Hou U, Nikos Mamoulis, and David W. Cheung. Efficient all top-k computation - a unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Trans. Knowl. Data Eng.*, 25(5):1015–1027, 2013.

[130] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 481–492, 2013.

[131] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, and Wenjie Zhang. Reverse k nearest neighbors queries and spatial reverse top-k queries. *VLDB J.*, 26(2):151–176, 2017.

[132] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, and Wenjie Zhang. Reverse k nearest neighbors queries and spatial reverse top-k queries. In *VLDB Journal*, 2016.

[133] Yunjun Gao, Xu Qin, Baihua Zheng, and Gang Chen. Efficient reverse top-k boolean spatial keyword queries on road networks. *IEEE Trans. Knowl. Data Eng.*, 27(5):1205–1218, 2015.

[134] Evangelos Dellis and Bernhard Seeger. Efficient computation of reverse skyline queries. *PVLDB*, pages 291–302, 2007.

[135] Yunjun Gao, Qing Liu, Baihua Zheng, Li Mou, Gang Chen, and Qing Li. On processing reverse k-skyband and ranked reverse skyline queries. *Inf. Sci.*, 293:11–34, 2015.

[136] Xiang Lian and Lei Chen. Reverse skyline search in uncertain databases. *ACM Trans. Database Syst.*, 35(1), 2010.

[137] Prasad M. Deshpande and Deepak Padmanabhan. Efficient reverse skyline retrieval with arbitrary non-metric similarity measures. In *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 319–330, 2011.

[138] Junchang Xin, Guoren Wang, Lei Chen, and Yunhao Liu. Energy-efficient reverse skyline query processing over wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(7):1259–1275, 2012.

[139] Xiaobing Wu, Yufei Tao, Raymond Chi-Wing Wong, Ling Ding, and Jeffrey Xu Yu. Finding the influence set through skylines. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 1030–1041, 2009.

[140] Md. Saiful Islam, Rui Zhou, and Chengfei Liu. On answering why-not questions in reverse skyline queries. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 973–984, 2013.

[141] Bugra Gedik, Kun-Lung Wu, Philip S. Yu, and Ling Liu. Motion adaptive indexing for moving continual queries over moving objects. In *CIKM*, 2004.

[142] Dmitri V. Kalashnikov, Sunil Prabhakar, and Susanne E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distributed and Parallel Databases*, 15(2):117–135, 2004.

[143] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD Conference*, pages 479–490, 2005.

[144] Kun-Lung Wu, Shyh-Kwei Chen, and Philip S. Yu. Incremental processing of continual range queries over moving objects. *IEEE Trans. Knowl. Data Eng.*, 18(11):1560–1575, 2006.

[145] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. Location-based spatial queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 443–454, 2003.

[146] Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 634–645, 2005.

[147] Xiaohui Yu, Ken Q. Pu, and Nick Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 631–642, 2005.

[148] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *2011 IEEE 27th International Conference on Data Engineering*, pages 541–552, April 2011.

[149] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 932–941, New York, NY, USA, 2012. ACM.

[150] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, and Ying Zhang. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. *VLDB Journal*, 2012.

[151] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD Conference*, pages 623–634, 2004.

[152] Xiaopeng Xiong, Mohamed F. Mokbel, and Walid G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.

[153] Muhammad Aamir Cheema, Yidong Yuan, and Xuemin Lin. Circulartrip: An effective algorithm for continuous nn queries. In *DASFAA*, pages 863–869, 2007.

[154] Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra. Dynamic queries over mobile objects. In *EDBT*, pages 269–286, 2002.

[155] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xuefei Li. Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *VLDB J.*, pages 69–95, 2012.

[156] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras, and Yufei Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *TKDE*, pages 1451–1464, 2005.

[157] Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Naixin Xu, and Andreas Züfle. Reverse k-nearest neighbor monitoring on mobile objects. In *GIS*, pages 494–497, 2010.

[158] Ioana Stanoi, Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Discovery of influence sets in frequently updated databases. *PVLDB*, 2001.

[159] Wei Wu, Fei Yang, Chee Yong Chan, and Kian-Lee Tan. FINCH: Evaluating reverse k-nearest-neighbor queries on location data. *PVLDB*, 2008.

[160] Muhammad Aamir Cheema, Zhitao Shen, Xuemin Lin, and Wenjie Zhang. A unified framework for efficiently processing ranking related queries. In *EDBT*, 2014.

[161] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[162] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *ICDE*, pages 189–200, 2010.

[163] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[164] Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 2002.

[165] Ahmad Abrishamchi, Ali Ebrahimian, Massoud Tajrishi, and Miguel A. Mariño. Case study: Application of multicriteria decision making to urban water supply. *Journal of Water Resources Planning and Management*, 131(4):326–335, 2005.

[166] RPA, Risk and Policy Analysts Ltd. Evaluating a multi-criteria analysis (MCA) methodology for application to flood management and coastal defence appraisals case studies report. *Joint Defra/EA Flood and Coastal Erosion Risk Management R&D Programme*, 2004.

[167] S. Patnaik, X.S. Yang, and K. Nakamatsu. *Nature-Inspired Computing and Optimization: Theory and Applications*. Modeling and Optimization in Science and Technologies. Springer International Publishing, 2017.

[168] Mingxi Wang, Shulin Liu, Shouyang Wang, and Kin Keung Lai. A weighted product method for bidding strategies in multi-attribute auctions. *J. Systems Science & Complexity*, 23(1):194–208, 2010.

[169] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.

[170] Nimrod Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.

[171] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.