CONTINUOUS SPATIAL QUERY PROCESSING IN MOBILE INFORMATION SYSTEMS

SHAULI SARMIN SUMI
Bachelor of Science, Islamic University, 2006
Master of Science, Islamic University, 2008

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

CONTINUOUS SPATIAL QUERY PROCESSING IN MOBILE INFORMATION SYSTEMS

SHAULI SARMIN SUMI

Date of Defence: December 07, 2017

Dr. Wendy Osborn Supervisor	Associate Professor	Ph.D.
Dr. Yllias Chali Committee Member	Professor	Ph.D.
Dr. Jacqueline Rice Committee Member	Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my parents.

Abstract

Nowadays, many mobile applications provide location-based services that allow users to access location-related data or information from anywhere, whenever they desire. A moving user can issue queries to access data or information about moving or static objects. Continuous spatial query processing systems are used for this type of application. Providing timely and useful location-based service is a challenging task because the results of the user query depend on the continuous updated location of the query issuer as well as on the location of the requested objects. We propose two query processing strategies for location-based services. The objectives of our strategies are to reduce: (1) the server workload, (2) the data transmission cost and (3) the query response time, for location-based services while providing an answer for a continuous region query. We found significant improvements in our strategies over the brute-force strategy. We compare our first strategy with a brute-force strategy and found that our strategy can significantly reduce the server workload and data transmission cost over the brute-force method. The data transmission cost and server workload of our second (improved) strategy are the same as the original strategy. We compare our improved strategy with the original strategy and brute-force strategy. The experimental results show that the improved strategy achieves lower query response time than the original and brute-force strategy.

Acknowledgments

Thanks to almighty Allah for giving me the energy, patient, and ability to carry out this thesis work.

This thesis would not have been possible without the guidance of my supervisor, Dr. Wendy Osborn. During my graduate study at University of Lethbridge, she has provided endless support. She always found time for discussing research, reading my texts, and giving valuable advices. I express my deepest gratitude to her.

I would like to thank my committee members Dr. Jacqueline Rice and Dr. Yllias Chali for their valuable suggestions during my thesis work.

I am thankful to the University of Lethbridge. I am also thankful to the School of Graduate Studies (SGS) for their financial support.

I must thank my parents and parents-in-law for their supports and prayers for me.

I would like to thank my daughter Sarah and my husband Asif. Thank you for all of your sacrifices, supports and motivation during my study, which has been enabled me to achieve this goal.

Lastly, I thank all other well-wishers, from whom I have got moral support, motivation and good wishes during my study in Canada.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Location-dependent queries [19] retrieve information based on the query issuer's current location. If the query issuer (i.e., client) is moving, and makes several requests for location information about some moving objects, then these types of queries are called continuous queries. Two types of continuous spatial queries are the continuous region or range query, and the continuous nearest-neighbor query. A continuous region query retrieves moving objects within a user-specified region based on the current location of a mobile user.

At present, many mobile applications provide location-based services [19] that make location-related information highly accessible anywhere. Moving users/clients issue the queries, requesting information to the server through a wireless network. The server processes these spatial queries and sends the answer to the user. In case of continuous queries, the results depend on the continuous updated location of the query issuer as well as on the location of the requested objects with respect to the updated queries.

1.1 Motivation

Location-based services provide various types of location-related information. One form of these location-based services is a continuous region query over moving objects or point sets where both the query issuer object and the target objects are moving. An example scenario: "A taxi driver asks for the locations of the customers within a radius of 2 miles who are looking for a taxi". The result of such query changes with the movement of objects and therefore requires continuous updates. The challenges of location-based services that

work with such types of continuous queries are [27]:

- Processing and sending the query results to the issuer requires a chain of server processes since the query issuer object is moving.
- The region of interest for the query is also changing due to the object(s) movement which adds more complexity to processing.

For a successful location-based service, the server workload and resulting data transmission cost should be minimized. Moreover, the server workload may affect and delay the overall query processing task. Any delay in query processing can make the query result outdated or useless since the objects are continuously changing their location. For example, in the previous scenario, if the query result is delayed, the taxi may pass out of the 2 miles region by that time, and then the results will be useless. So, to make the location-based service efficient, the delay in query processing needs to be minimized.

In a simple strategy for processing continuous range or region queries, each object reports its location as it moves and changes location. The server updates the query results according to the changed locations. In another strategy, the server monitors the location of both the objects and the queries, and continuously updates the query results as the clients/user and objects move.

In the processing of continuous queries, various research works have been undertaken. However, the ever increasing use of mobile devices and the continuous development of wireless networks make continuous location-depended queries an important focus of research. IIarri *et al.* [18] propose a system that supports distributed processing of continuous location dependent queries in a mobile environment, using mobile agents to carry out the necessary processing tasks. Liu and Hua [28] propose a distributed framework to process moving queries, such as the moving range query and the moving kNN query, over moving objects in a spatial network environment. Lin and Wu [26] propose a system architecture for a road network, and also provide an approach to determine the safe period

for updating the query result for both the centralized and distributed processing of directional continuous range queries. Their experimental results show that the average query response time is shorter for centralized systems than for distributed systems. In the case of distributed query processing systems, a lot of computation and communication among access points is needed, and thus the response time for each query increases with the increased number of mobile objects. Gupta *et al.* [8] propose a hierarchical database framework for location-dependent query processing in a mobile environment. This framework is based on a tree architecture and the authors propose an algorithm to support this proposed tree architecture. Wang and Zimmermann [39] propose a method to process Continuous Location-Based Range Queries on moving objects in road networks. For this method, they present a dual-index structure and a precomputing component to facilitate the query processing. The authors also present a data structure called the Shortest-Distance-based Tree (SD-Tree) in order to reduce the query update cost. They also propose a Continuous Mobile Network-Distance-based Range query algorithm (C-MNDR) to process a continuous range query. Moreau and Osborn [29] propose a two-dimensional spatial access method called the mqr-tree. They also present their insertion and region search algorithm. Their experimental evaluation shows that the mqr-tree region search performance is better than the benchmark indexing strategy in terms of the number of disk accesses required for performing the search process. We present further detailed discussion on existing research work for continuous queries in Chapter 2.

Most of the existing works on continuous range queries over moving point data do not use a spatial access method. Some existing works use spatial access method which contain overlapping. But the existence of overlapping can affect the query processing/response time due to the unnecessary searching in the overlapped region. Moreover, validity regions for a query are not obtained directly from those spatial indices which adds extra computational cost. Other existing works, especially in broadcast networks, use monitoring for continuous location updates, but this monitoring is performed by the client and not the server, and

therefore can affect the response time of a query.

To address these issues, we propose two efficient processing strategies for continuous region queries over moving point sets where both the query issuer object (i.e., client) and the target objects are moving. Our goal is to reduce the server workload, result/data transmission cost and the query response time. In our environment, moving users issue the queries, requesting information from a central server through a wireless network. The server will use the mqr-tree spatial access method, including the mqr-tree region search, for query processing. The server processes the spatial queries and sends the answer to the user. The result of the continuous query changes with the movement of the objects, and therefore requires continuous updates. The server will also identify the changes of the object(s) location for which the query results will remain unchanged, in order not to send the query result for any location changes. Therefore this reduces the server workload, speeds up the overall response time, and reduces data transmission cost.

1.2 Overview of our work

We propose two query processing strategies for location-based services, the goal of which is to reduce the server workload, data transmission cost, and query response time. Both strategies are for continuous region query processing over moving point sets, where the query issuer object or client is also moving. In our environment, the client issues region queries to a central server and the server processes the query and sends a result back to the client only if the current result is different from the previous query result. In our first strategy, the server compares the current result set with the previous result in order to identify any change in the new result. If the current result is different from the previous query result, the server sends the entire result file. Otherwise it sends a message indicating that the current result is the same as the previous. Our proposed strategy does not send the entire result file for every query in order to reduce the data transmission cost and sending workload. In some cases for the first strategy, when the result contains a large data set, then

the data comparison between large result sets causes significant overhead for the server. The comparison time of the first strategy also increases the query response time. In order to avoid this problem, we propose an improved strategy which monitors and tracks the points that have moved, entered or have gone outside the query region. Instead of comparing all points in the result set, our improved strategy tests only the moved point(s) to make a decision as to whether the current result is different or not from the previous query result. This reduces the result comparison overhead of the server and at the same time reduces the query response time.

1.3 Contributions

Our contributions in this thesis are as follows:

- A spatial indexing [38] mechanism that is non-overlapping for point data is used to speed up the searching process. We apply the mqr-tree spatial access method to solve the problem of moving range/window queries on moving point data. The mqr-tree [29] is a two-dimensional index structure that organizes spatial objects in a two-dimensional node based on the spatial relationship. For point data, the mqr-tree contains no overlap and therefore the search process for point data will proceed along one path which makes the search process and data retrieval efficient.

- We obtain a validity region (i.e., in our work, a superMBR) from the spatial access method itself. It is possible because of no existing overlap of the minimum bounding rectangles (MBRs) for point data in the mqr-tree [29], which is not possible in the case of other spatial access methods.

- In our proposed improved strategy, we use a server-side monitoring technique which is more efficient than other existing works.

1.4 Organization of the thesis

The rest of this thesis is organized in the following order. Chapter 2 provides the related work for continuous spatial query processing in mobile information systems. Chapter 3 presents our main contributions. Our proposed original strategy and the improved strategy are described in this chapter. In Chapter 4, we present the implementation details and the experimental results of our strategies. Finally, Chapter 5 presents the conclusion of our thesis work and also provides future research directions.

Chapter 2

Background

Nowadays, there are many mobile applications that allow users to access data or information from anywhere at anytime [19]. The data or information can be about moving or static objects. A user who is also moving or stationary can issue queries to access these data. For example, they may search for the locations of particular object(s) which can be either static or moving. Continuous spatial query processing systems are used for this type of application.

Two types of continuous spatial queries are [19]:

1. Continuous region query

2. Continuous nearest-neighbor query

The continuous range query involves finding information of moving objects within a user specified region and continuously monitoring and updating the results as the user and/or the objects move. If this region is a rectangle, then the query is also called the continuous region or window query. An example of continuous range query can be: *A traveller while walking in a new city, issues a query to know the locations of all hotels or taxis within 100 meter region around him/her*.

The continuous nearest-neighbor query involves finding one or more nearest neighbor(s) (kNN) of a moving user and updating as the user's location changes. An example of a continuous nearest-neighbor query is: *A mobile user issues a query in order to know the location of the nearest taxi while he/she is walking towards home*.

In this chapter we discuss existing work on continuous spatial query processing approaches. In Section 2.1, we describe several existing works on processing continuous range queries and window queries. Approaches for processing continuous nearest neighbor queries are described in Section 2.2. Section 2.3 describes some existing approaches that are designed for processing both range and nearest neighbor queries.

2.1 Continuous range queries and window queries

A continuous range query retrieves moving objects within a certain region. If the region is a rectangle, then it is called a region or window query. This section describes various existing approaches related to the continuous range and window queries.

2.1.1 Distributed Processing Using Mobile Agents

IIarri *et al.* [18] propose a system that supports distributed processing of continuous location dependent queries in a mobile environment, using mobile agents to carry out the necessary processing tasks.

The authors argue that most of the previous proposed works for continuous moving queries are based on centralized processing. However, it is not always feasible for a centralized system to keep track of all moving objects because there could be millions of moving objects in a large area. The authors identified one work that distributes the query processing but this approach can overload wireless devices. They conclude by saying that no other general distributed architecture exists that can be easily adapted to processing a continuous location-based query.

Their approach uses an overall area that is divided into *proxy areas*. The location information of moving objects within a *proxy area* is managed by a Data Management System (DMS) on a fixed set of computers called a *proxy*. Therefore, the query processing task is distributed among the different *proxies*. According to the authors, such a distributed query processing task provides the following benefits [18]:

- Query processing is performed on multiple servers, eliminating the overload on mobile devices.
- Query processing for different geographic areas can be performed in parallel.
- New *proxy areas* can be added easily.
- Service availability is improved.

The authors propose a dynamic tracking architecture, where mobile agents are used to track interesting moving objects. Results are updated at a certain frequency, which is specified by the user.

The user device or computer is called a monitor, which creates *QueryMonitor* agents to answer queries and keep the retrieved data up to date. The query is processed following the steps shown in Figure 2.1[18]:

- The user provides their query to the processor in a natural language. A natural language processor is used to transform the location query into an SQL-like format (Step 0). An example query [18] is: *Give me the police units within 0.56 miles around car38*.
- The user query is analysed to identify the *reference objects*, the *target object* and the *relevant area*. For the above example, the *reference object* is the car38, the *target object* is the police units and the *relevant area* is the 0.56 miles region around the car38. An extended area for each *relevant area* is also identified by considering the maximum speed of the *relevant object* and the answer refreshment frequency (Step 1).
- The SQL-like query is converted into a standard query (Step 2).
- A network of agents is arranged to perform the query processing task in a distributed way (Step 3).

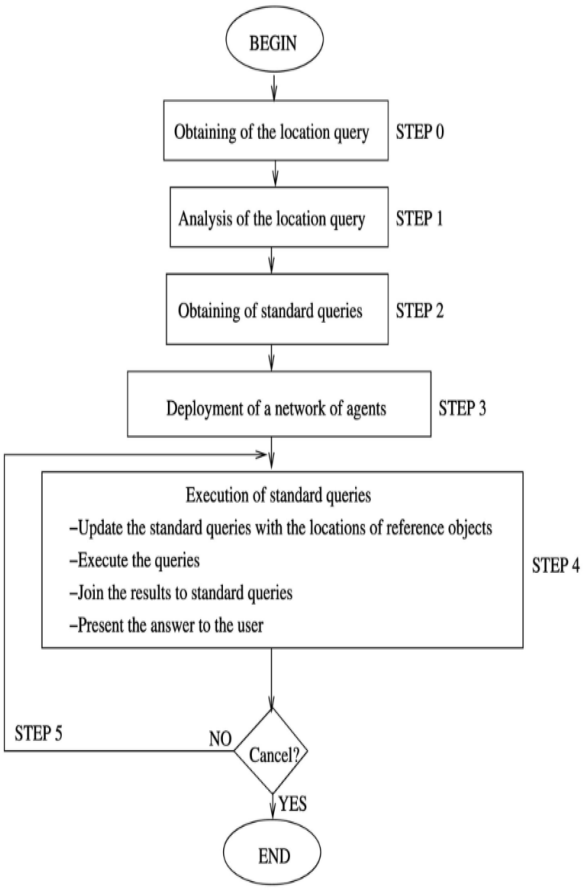


Figure 2.1: Location query processing steps (from [18])

- The standard query is executed to obtain the *target object* from DBMs. The *Query-Monitor* collects the results and presents them to the user. In addition, the results are refreshed at the user specified frequency (Step 4).

The authors perform some experiments to show the behavior of their system. They evaluate the precision and scalability of the system. A simple location estimator is used to approximate the current location of an object by using its previous location, until the actual location is received. The experiments assume a refreshment period of 5 seconds, and a worst case scenario where the proxies have a small coverage area.

The first precision experiment considers a situation of tracking the location of two cars of varying speed. Results show that minimum imprecision occurs at every 5 seconds, but the moving object continues moving within that time. In addition, it is found that: (1)

imprecision occurs when the objects move between proxy areas, and (2) errors increase as the objects move faster. Another observation shows that sometimes extra objects are presented to the user, and some objects do not appear that should be present. The third precision experiment considers different speeds and refreshment periods. It is found that when the speed of the moving objects increases, the precision error increases, but for very frequent refreshment periods, the precision error remains lower even for higher speeds.

The first scalability test involves increasing the number of concurrent queries. The result shows that: (1) the average location error committed by the system increases very slightly when the number of queries increases, and (2) both the CPU and memory usage increase only slightly as the number of queries increases. The second scalability test involves changing the numbers of objects in a scenario. The results show that the average location error detected by the proxies is slightly increased as the number of objects increases. The authors conclude that with their distributed infrastructure, location tracking can be done more efficiently, even with a large number of moving objects, because each proxy handles a much smaller number of moving objects than it would with a centralized approach.

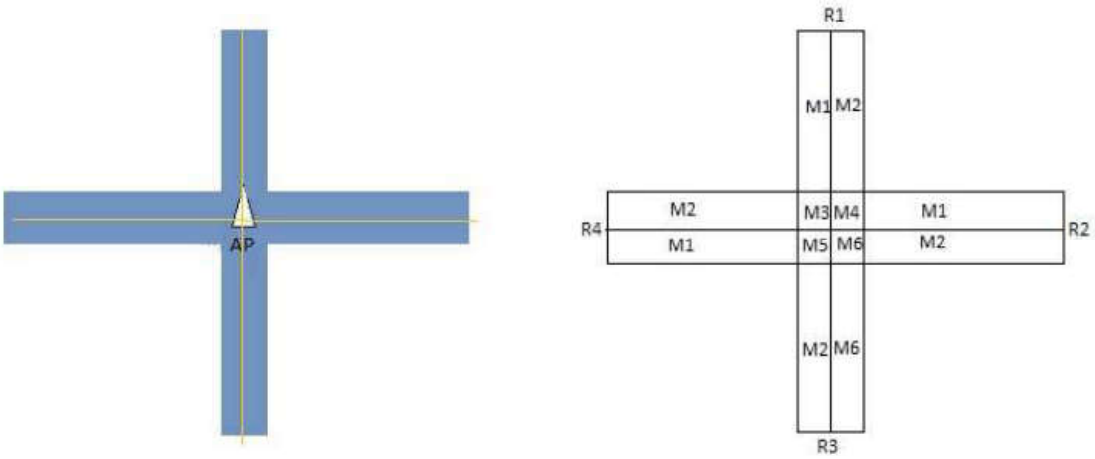
2.1.2 Directional Continuous Range Queries on Road Networks

Lin and Wu [26] propose a system architecture for a road network, and also provide an approach to determine the *safe period* for updating the query result for both the centralized and distributed processing of directional continuous range queries (e.g., *How many taxis will appear on the way to my destination?*).

Several existing studies of query processing on road networks assume the use of omnidirectional queries. However according to the authors, the direction of the traveller, which is ignored in an omnidirectional query processing system, has a significant influence on the result of the query.

In their proposed architecture, the authors assume that each mobile object (e.g., car) is equipped with a Global Positioning System (GPS) and its travel direction is predetermined

by a central server, which includes a path planning system such as Google Maps. At each intersection of the road, there is an access point. All access points in a road area are interconnected by underground cable to form a mesh network, which is connected to the central server. The access points can monitor and communicate with a mobile object within a fixed range by wireless communication. The task of an access point is to keep information about the mobile objects, processing query, and relay messages between mobile objects and the central server.



(a) Roadlet of 4-way intersection (b) An intersection and four road section

Figure 2.2: Representation of a roadlet (from [26])

This system divides a road into *roadlets* as shown in Figure 2.2. A *roadlet* consists of an intersection and the road sections that are connected to the intersection. A *roadlet* has a unique access point ID and is divided into five regions:(i) the intersection, (ii) the eastern road section, (iii) the western road section, (iv) the southern road section, and (v) the northern road section. An intersection is divided into four equal blocks: (i) the upper right, (ii) the upper left, (iii) the lower left, and (iv) the lower right, as shown in Figure 2.2(b). Based on driving direction, each road section is divided into two blocks. A road section is identified by a unique road block code, which is generated by sequentially concatenating the access point ID, the region ID, and the block ID. Finally the travel path for the query object is represented as a sequence of road block codes. The first element of a road block

code is the current location of the mobile object, and with the movements of the mobile object, the code is updated.

At any particular time there may be many mobile objects on a road. The proposed system processes a directional continuous query by comparing the road block code of the query mobile object with the code of every other mobile object on the roads, to see if there exists an intersection between any two road block codes. If there exists no intersection, no mobile objects will be part of the query result, and therefore, there is no need to monitor other mobile objects. But if an intersection exists, some mobile objects will be part of the query result, and so those mobile objects will need to be monitored. Generally more accurate query results can be achieved if the current location of the mobile object can be updated frequently. However this requires more computational overhead and more network bandwidth. To solve this problem the authors also provide an approach to determine the *safe period* for updating a query result instead of using frequent updating.

The *safe period* for updating a query result is the time when it is necessary to perform update. The query result must be updated in two situations: (1) when any objects have left the query region (2) when any new objects enters the query region. Both scenarios define safe periods for updating the query result.

The authors perform some experiments to evaluate the performance of both the centralized and distributed continuous range query processing over their proposed architecture. They perform their experiment in a 1.6km X 1.6km area of Tainan city in Taiwan, which contains seventy three intersections and over one thousand road blocks. The authors evaluate the average response times by varying the number of mobile objects and the number of queries. Results show that the average response times of the distributed system are shorter than for the centralized system for 300 or less mobile objects. When the number of mobile objects increases to 500 or above, the average response times of the distributed system become larger than the centralized system and increase rapidly. For 100 or less queries, the response times of the centralized and distributed processing systems are almost the

same. When the number of queries increase to 300 or above, the average response times for both systems increase, and for the centralized system, it remains lower than that of the distributed system. The authors conclude that in the case of the distributed query processing system, significant computation and communication among the access points is needed, and therefore the response time for each query increases with the increased number of mobile objects.

2.1.3 Other Approaches

Wang and Zimmermann [39] propose a method to process continuous location-based range queries on moving objects in road networks. According to the authors, the previously existing approaches do not support the computation of network-based distance and the capability to process moving objects as a point of interest, simultaneously. They present a dual-index structure and a precomputing component to facilitate query processing. The authors present a data structure called the Shortest-Distance-based Tree (SD-Tree) in order to reduce the query update cost. They also propose a Continuous Mobile Network-Distance-based Range query algorithm (C-MNDR) to process a continuous range query. The authors perform several experiments to evaluate the performance of their method, and the results show that the method performs very well even in a road network with a very large number of moving objects.

2.2 Continuous nearest neighbor queries

Continuous k-nearest neighbor (CkNN) queries retrieve the moving objects that are closest to a certain object or location along the entire path from its source to destination (e.g., finding the three nearest restaurants to a moving car on any point of a pre-specified path) [21]. The outcome of this type of query is a set of intervals or split points and their associated kNNs. An interval is defined where the kNNs of all points is the same. On the path, a split point is defined where the kNNs of a moving object change. There are various

approaches that deal with the problem of processing continuous nearest neighbor queries. Some of these approaches are summarized in this section.

2.2.1 Continuous K-Nearest Neighbor Queries in Spatial Network Databases

In a spatial network database (SNDB) (e.g., road network), the objects move in a predefined path. The distance between two objects (e.g., a vehicle and a hospital) is computed as a function of the network connectivity, or the shortest path between two objects. Kolahdouzan and Shahabi [21] propose two techniques for processing CkNN in a SNDB.

According to the authors, existing CkNN strategies are based on the ideas proposed for regular kNN queries, including the use of an index structure, which work by using the filter and refinement steps. The main drawback of these algorithms is that in the filter step, an Euclidean distance metric is used to find the distance between two objects. Therefore the exact distances in spatial networks cannot be calculated.

The authors focus on the situation where more than one neighbor (i.e., k $>$1) is requested by a moving query object. Two approaches are proposed: the Intersection Examination (IE) approach and the Upper Bound Algorithm (UBA). In IE, the entire path is divided into smaller segments. A segment is the path between two adjacent nodes in the path. Thus, each segment is surrounded by two adjacent nodes. The kNNs of the two nodes of each segment are identified. Using the kNNs of the nodes, the location and the kNNs of the split points on each segment are identified. Finally the union of the kNNs of all split points along the path is identified as the resulting kNNs of the moving query object. IE performs kNN queries for every node (e.g., split point) on the path, although when an object moves slightly, its kNNs will generally remain unchanged. An issue with the IE strategy is that this huge computation may be unnecessary.

In the UBA approach, the kNNs for only a subset of the nodes on the path are computed. This is done by calculating a minimum distance between subsequent nearest neighbors of an object, within which the query object can move without requiring a new kNN query. Again,

the UBA determines the adjacent nodes that cannot have any split points in between, and does not compute the kNN queries for those nodes. In this way, the UBA approach reduces the number of kNN computations in comparison with the IE approach.

The authors perform some experiments to evaluate and compare the performance of their proposed approaches for different values of k, and different path lengths. They use real world data sets. One data set represents a network of approximately 110,000 links and 79,800 nodes of the road system in downtown Los Angeles. In addition, different sets of points of interest are used (e.g., hospitals, restaurants, etc.). The authors find that for 1NN, the query response time is almost instantaneous for all sets of points of interest, regardless of density. In addition, the query response time of UBA is always less than that of IE, regardless of the k value or path length. Finally, the performance of both approaches increases linearly when the path length increases.

2.2.2 Sliding Windows

Mouratidis and Papadias [31] address the monitoring of continuous kNN queries over sliding windows. In the sliding window model, data points arrive continually but only some most recent data points are used during query processing. It can be *count-based* or *time-based*. The *count-based* sliding window contains the W most recently arrived data points, and the *time-based* sliding window contains the data points that arrive within a fixed time interval W. The authors mention that some of the previous works focus on continuous monitoring of NN queries where an object issues an update only when it changes its position, and the server processes the stream of position updates and incrementally maintains the NNs for the query. They state that no work has been done on continuous monitoring of NN queries over sliding window.

The authors propose two monitoring approaches for continuous kNN queries over sliding windows: the Conceptual Partitioning (CPM) algorithm [32] adapted to the sliding window model, and the Skyband Nearest Neighbor (SNN) algorithm, which is based on

the concept of a k-skyband [35]. Both algorithms work for count-based and time-based windows.

In their work, each data item is considered valid only while it resides in a sliding window. A single list stores all the valid points. For both count-based and time-based versions of the sliding window, the data points are added and removed in a first-in, first-out manner. Therefore, the window contains only the most recent data points. In both of the proposed approaches, the initial kNN set for a query is computed using the CPM algorithm. When any insertion or deletion of data points occurs in a window, only then the current kNN set changes. For the SNN approach, the current results are represented in the form of a k-skyband. A k-skyband representation contains the nearest neighbors that are dominated by at most (k - 1) other ones in the distance-time space [31].

The authors perform some experiments to evaluate the performance of their proposed approaches. They use a real data set containing the end points of streets in Los Angeles. Their experiment results show that the SNN is faster because it requires fewer recomputations of the query. The experiment also shows that the SNN requires lower CPU cost but only a few kilobytes of extra memory space than CPM. The authors conclude that the SNN outperforms the CPM but uses extra memory space which is very small in amount and therefore negligible.

2.2.3 Moving Objects with Uncertain Velocity

Huang *et al.* [13] propose a Probability-based Possible-KNN (P2KNN) algorithm to efficiently process a CkNN.

According to the authors, reducing the overhead of performing repetitive queries is a major problem for efficiently processing a CkNN query. The authors identify the following limitations in previous works:

- The velocity of a moving object is fixed. Every time the object changes its velocity, the CkNN query needs to be executed again, which results in a high re-execution cost

when the moving object frequently changes its velocity.

- The velocity of the moving object can vary, and a snapshot kNN query can be re-evaluated only when updates occur, but the query re-evaluation cost is still high.

Therefore, the authors propose an efficient method that determines the kNNs for each time instant by evaluating the query only once. This method assumes that the velocity of an object is uncertain but varies within a known range. The distance between a moving object and the query object becomes uncertain too. The authors first propose an uncertain distance model to calculate this uncertain distance between moving objects at each time instant. The Probability-based Possible-KNN (P2KNN) algorithm finds the possible kNNs of the query object at every time instant within the user specified time interval. This algorithm works in three phases: (1) Pruning phase, (2) Candidate-distilling phase, and (3) Ranking phase. In the pruning phase, the objects that are impossible to be in the query result are removed from consideration based on a pruning distance. In the candidate-distilling phase, the subintervals are identified where the query object has the same possible kNNs within each subinterval. Then the objects that are the probable kNNs of the query object at each time instant are determined. In the ranking phase, a confidence value is calculated based on the probability for each possible kNN of the query object. Then the most appropriate kNN is chosen by choosing the maximum confidence value.

The authors performed some simulation-based experiments to evaluate the performance of their proposed approach. One synthetic data set and one real data set are used. First, they investigate the precision of their algorithm by identifying the actual number of kNNs that are retrieved through the ranking phase. The experiments show that for increasing values of k, the precision of the algorithm is above 85% at earlier time instants. As time passes, the precision decreases, but remains over 70%. The authors conclude that the ranking mechanism identifies the most appropriate kNNs. Next, they evaluated the algorithm performance by measuring the average CkNN query processing time with a varying number of moving objects, nearest neighbors, and uncertain velocity range of the moving object. Results show

that the overall query processing time is low, and is insensitive to the query interval length, the number of objects, the value of k, and the uncertain velocity range of the moving object. Finally, the authors compare the performance of the P2KNN algorithm with other CkNN approaches. The authors conclude that their approach is more suitable in a highly dynamic environment where the object velocity changes frequently.

2.2.4 Speed and Direction of moving objects in road network

Fan *et al.* [4] propose a Moving State of Object (MSO) model to determine the relative moving situation of an object towards the query point. Using this model, they propose an Object Candidate Processing (OCP) algorithm in order to reduce the overhead of executing repetitive queries.

According to the authors, existing approaches for CkNN queries have the following limitations: (1) Some are based on euclidean spaces, but in road networks, the distance between a data object and a query point is the length of the shortest path connecting them (2) Some assume a static object set, and (3) Others have data objects and query points that move continuously in a road network, but they require much communication cost to process the query.

This approach considers the moving state of an object, which indicates whether an object (from the set of kNN of the query point) is moving away from or getting closer to the query point. The MSO model examines the relative moving speed and direction of the object toward the query point. The OCP algorithm has two phases: (1) pruning, and (2) refining. In the pruning phase, the objects which cannot be in the CkNN query results within a given time interval are pruned or ignored, based on a pruning distance. Here, the pruning distances are kept smaller so that the number of candidate objects remains small. The CkNN query results may change during the given time interval due to the frequent updates of the object location. In the refining phase, the time subintervals are determined, where a certain query result can be obtained. Here, some objects which cannot change the

kNN query result are excluded.

The authors perform some experiments to evaluate the performance of their proposed OCP approach and also compare it to two existing algorithms: the CKNN algorithm [14] and the Incremental Monitoring Algorithm (IMA) [33]. A synthetic data set and two real data sets are used. In the experiment, the objects start moving from a randomly selected position and pick a random direction and move at a speed between 0 mph and 70 mph. The authors set the default k value to 10, and the length of the query time interval to 100 time units for the experiments.

The authors evaluate the effect of the query time interval length, the number of nearest neighbors, and the number of query points on both the precision and CPU cost. It is found that the running cost increases with the increase of the time interval length, the number of nearest neighbors, and the number of query points, for all data sets. In addition, OCP performs better than the other two algorithm with less running cost.

It is also found that the precision of the OCP algorithm remains about 100% with the increase of the query interval length and the number of nearest neighbors. The precision also reaches above 95% with the increased number of query points.

2.2.5 Other approaches

Gao and Zheng [5] introduce a new type of continuous nearest neighbor query, in an obstructed space called a Continuous Obstructed Nearest Neighbor (CONN) queries. According to the authors, existing approaches for CNN search do not consider the impact of obstacles on the distance between objects. The CONN query processing algorithm searches for nearest neighbors by considering the impact of obstacles on the distance between objects. Gao *et al.* [6] extend the CONN approach to process two variations of CONN queries, called the Continuous obstructed k nearest-neighbor (COkNN) search and the Trajectory obstructed k nearest-neighbor (TOkNN) search. The COkNN search approach retrieves the k obstructed nearest-neighbors of every point on a given query line segment. The TOkNN

search approach finds the k obstructed nearest-neighbor for every point along a specified query trajectory consisting of several consecutive line segments. The authors also propose an approach for approximate COkNN (ACOkNN) search. The authors perform several experiments to evaluate the effectiveness and efficiency of their proposed approaches.

Gao *et al.* [7] also consider the impact of obstacles on the visibility between objects. According to the authors, existing continuous nearest neighbor query processing approaches do not consider the impact of obstacles on the visibility between objects. They introduce a new type of spatial query called the Continuous Visible Nearest Neighbor (CVNN) query. They also introduce several variations of CVNN queries: (1) CVkNN search, which retrieves the k VNNs for every point on a given query line segment (2) TVkNN search, which returns the k VNNs of every point along an arbitrary trajectory (3) δ-CVkNN search, which consider a visible distance threshold δ, and (4) CCVkNN search, which retrieves the k VNNs in the restricted area (defined by the spatial region constraints) for each point along a specified query line segment. To improve the search performance, the authors present a series of pruning heuristics on the data set and obstacle set respectively, which are used to prune the objects and obstacles that have no impact on the final query results. They perform several experiments to verify the effectiveness of their approach.

Kwon *et al.* [23] propose a distributed continuous kNN query monitoring method for a hybrid wireless network, where moving objects construct the ad-hoc network in the wireless broadcasting system in order to reduce the communication cost. In this system, each moving object has two radio interfaces: one for WiFi and the other for cellular networks. The WiFi links are used together with a few cellular links to send the current location of the moving object to the server in order to minimize the burden on the cellular uplink communication cost. According to the authors, the existing approaches uses many cellular wireless connections, which increases the cellular communication costs for the overall query processing system. The authors perform several experiments to evaluate the performance and the results show that their proposed method performs better than other existing approaches

in terms of communication cost.

2.3 Continuous range and nearest neighbor queries

Some proposals have focused on the processing of both the continuous range/window queries and continuous nearest neighbor queries. This section describes some of these works.

2.3.1 Location-Based Spatial Query Processing in Wireless Broadcast Environments

Ku *et al.* [22] present a query processing technique that maintains high scalability and accuracy, and is also able to reduce the delay in answering the query. It uses peer-to-peer sharing of cached query results with its neighboring mobile peers.

The authors state several limitations of existing approaches for ad-hoc queries [22]:

1. Not suitable for a very large number of users.
2. The need to communicate with a server via a fee-based cellular-type network.
3. The need to send location information to the server, which is not desirable for privacy reasons.

Several of these limitations are addressed in a wireless broadcast approach, where the server repeatedly broadcasts information in a wireless channel, and the mobile clients tune in to the channel for the information that they desire. The authors mentioned the drawback of this model is that it has a high latency, as clients need to wait for their desired information in a broadcast cycle.

To eliminate these limitations, the authors propose a sharing-based approach in an ad-hoc broadcat network, for location-based spatial query processing (i.e., the kNN query and the window query). Ad hoc broadcast networks are used to share information among mobile clients in a peer-to-peer manner to reduce the latency. For their method, cached results of spatial queries are shared among reachable mobile hosts, to be used by them for answering

future queries. The authors assume an infrastructure where mobile clients are vehicles, each of which is equipped with a GPS that provides continuous position information. However, their approach also works for other mobile devices. They also assumed that the wireless server transmits information periodically. There also exists a short-range network for ad-hoc communication among neighboring mobile clients. Each mobile client has sufficient transmission range and a virtually unlimited power lifetime. When a mobile host needs an answer to a spatial query, it obtains data from the broadcast channel, and also from the caches of neighboring peers.

The authors also propose a set of verification algorithms, which verify whether the data collected from a neighboring peers is complete, partial, or irrelevant to answer a spatial query. If the verified result contains only part of the query answer, the query issuer needs to wait for the necessary data from the broadcast network to obtain the remaining answer. In many situations, the partial answer can be used by many mobile hosts that prefer an approximate result that arrives in a short time, rather than an exact solution with a long latency. With this approach, the access latency is reduced because many queries are answered directly by peers. The remaining queries that can not be answered by peers, are solved by obtaining the required data from the broadcast channel.

The authors evaluate the performance of their proposed approach in terms of two issues: (1) what percentage of the client spatial queries are answered directly by peers, and (2) how many broadcast data packet are required compared with the existing approaches to answer a sequence of queries with partial results from a sharing-based approach.

For the experiments, the authors use two real-world data sets that represent high and low density areas, and a synthetic data set that represents the vehicles and object densities in between the two areas. To evaluate the performance for kNN queries, the authors observe the percentage of total solved queries by peers, by varying the wireless transmission range, the cache capacity, and the number of nearest neighbors. Results show that with an extended transmission range, an increasing number of queries can be answered by the surrounding

peers. They also observe that: (1) with a higher mobile host cache capacity, a large number of queries can be solved by the peers, and (2) their approach is more effective when the number of nearest neighbors is small. To evaluate the performance for window queries, they also vary the wireless transmission range, the cache capacity, and the query window size. Also for window queries, with extended transmission range and higher cache capacity more queries can be solved by the surrounding peers. In the experiments to evaluate the second issue, the results show that their approach can reduce the wireless broadcast channel access, by a significant amount (e.g., up to 80 percent) in a dense area. From the overall experiments, the authors found that if more mobile clients travel in an area, then the higher the number of spatial queries that are possible to be solved by peers with their approach and consequently, the access latency is reduced significantly because of less access to the wireless channel.

2.3.2 Continuous Monitoring of Spatial Queries in Wireless Broadcast Environments

Mouratidis *et al.* [30] propose an air indexing framework for continuous spatial query processing in a wireless broadcast environment, in order to reduce the power consumption and access latency at the client side.

The authors state some limitation of existing approaches:

- Some are designed for snapshot queries over static data and are not suitable for continuous queries with dynamic data.
- Others use a central server system where with the increased numbers of queries, the processing load at the server increases and it takes a longer time to answer the queries.

The authors propose a Broadcast Grid Index (BGI) method. It is designed for both ad-hoc and continuous queries over both static and dynamic data. In this method, the data are indexed in a regular grid, where the data space is partitioned into square cells of equal size, with a side-length defined by a system parameter. Each cell stores the object coordinates

that fall inside of it, and also stores the total number of objects in the cell. The small grid structure of the BGI method is efficient in broadcasting environments, and also allows for fast object updates so that server overloading is avoided in the situation of numerous updates.

For the ad-hoc and continuous kNN queries over static data, the index information is broadcast to all clients in two parts. The first part contains the cell cardinalities and the second part contains the object coordinates of each cell. But for continuous kNN and range queries, clients have to continuously monitor their results as they move. To make this query processing more efficient, the authors propose an alternative monitoring algorithm, which uses the previous query result to reduce the overhead. The server broadcasts a dirty grid at the beginning of a cycle. A dirty grid cell contains only one bit which is initiated to 0, instead of objects and cardinality information. When any object is inserted or deleted inside a grid cell region, then the bit is updated to 1, and this cell is said to be dirty. This dirty grid indicates that some data space regions have received updates during the cycle. The clients need to re-execute their query only if the affected regions invalidate their current result.

The authors evaluate the efficiency of their proposed algorithm for window and kNN queries by an experimental comparison with existing state-of-the-art frameworks. They also compare their algorithm (with monitoring) for continuous queries with the naive constant re-computation technique. They measure the efficeincy in terms of two performance metrics [30]: (1) tuning time (i.e., the total time that the client stays in the receiving mode to process the query), and (2) access latency (i.e., the total time elapsed from the moment the query is issued until the moment that all the corresponding objects are retrieved) by varying different parameters (e.g., database size, packet size, range query area, number of NNs, object/query speed and object/query movement).

Both real and skewed spatial data sets are used. Results show that the tuning time and access latency for BGI are lower than the existing state-of-the-art methods. In addition, the authors found that with respect to tuning time and access latency, BGI is much better

than re-computing for both static and dynamic data, because the monitoring method reuses the previous query results. However, for moving object queries, the tuning time of BGI increases with the object movement, because more queries need to be re-evaluated per time stamp and monitoring becomes worse than re-computation, which remains unaffected by the object movements.

2.3.3 Indexing for Location-Dependent Spatial Queries

Park [36] proposes an approach for mobile peer-to-peer (MP2P) services in wireless broadcast environments, using a new Hierarchical Location-Based Sequential (HLBS) index that provides selective searching entries without pointers based on the location of each data object. The author also proposes a new data dissemination algorithm called the Nearest First Broadcast (NFB) algorithm.

Each node of a MP2P system consists of low battery power, a slow processor, and limited memory. According to the author, conserving battery power, speeding up query processing, and protecting user personal information are vital for improving location-based services on mobile devices. The author identifies some limitation of existing MP2P system:

- The current location information of a mobile user must be delivered over the network while processing a query, which is vulnerable to security attacks.
- Many of the existing approaches that use index structures and algorithms are designed for traditional spatial databases and cannot be easily applied for spatial queries in wireless data broadcast environments.

In a MP2P scheme, a node can act as a service provider as well as a client. Based on the current location, the service provider periodically broadcasts the location-dependent data items. In the proposed approach, the HLBS index tree is designed based on a minimum bounding rectangle (MBR). This index consists of three levels: (1) root-MBR (RMBR), (2) sub-MBR (SMBR), and (3) leaf nodes. At each level, an internal node contains sequentially

ordered information based on its location instead of pointers for a lower level node. During query processing, a client starts searching from the RMBR, and then sequentially checks each node of the same level, according to the broadcast sequence, to identify the node which is required for the result. The search traverses down to the lower level and continues until the required leaf node that points to the corresponding object is found.

In NBF, the IDs and the coordinates of the data objects with other information (e.g., images and text) are periodically broadcasted by a node to the clients. The data objects are sorted sequentially using the NBF algorithm before they are broadcasted to the clients. Each node broadcasts the closest data object first, and then sequentially broadcasts the next closest data object. The algorithm allows clients to tune selectively using a linear accessing structure based on the location of each data object. This can also protect the leakage of personal information because the client is not required to disclose their private location information in the broadcast channel.

The author conducts a simulation-based experiment to evaluate the performance of the proposed algorithms. A synthetic dataset and two real datasets are used. First, the author compares the average access time and tuning time of HLBS with traditional spatial indexing algorithms such as the R-tree [9] and the Distributed Spatial Index (DSI) [40] techniques by varying the percentage of the service area and the size of the range query. Results show that the performance of HLBS is better than both the R-tree and DSI in term of access time and energy consumption. The small size and sequential structure of the HLBS reduces the access latency and also increases the search efficiency. The author compares the NFB performance with the BBS [37] approach and the random broadcast technique in which information is transferred through a wireless data channel in response to a client request. Results shows that the NFB outperforms BBS and the random broadcast approach since the client can obtain the desired data in a minimal time with the NFB algorithm. Finally the author compares the performance of three combinations: HLBS with NFB, R-tree with NFB, and HLBS with BBS. Results show that HBLS with NFB outperforms the other two

combinations, because both the index search cost and the data access time remain low.

2.3.4 Other approaches

Huang and Huang [12] propose a proxy-based approach for continuous NN and window queries in a mobile environment, in order to reduce the number of queries from the clients, the latency in achieving the query results with associated estimated valid regions (EVR), and the server load. They also present algorithms to efficiently create larger estimated valid regions (EVR) for NN and window queries on static data objects for mobile clients. Finally, the authors conduct several experiments and the results show that their approaches perform much better than an existing proxy-based approach by creating larger EVRs for mobile clients.

Lam and Ulusoy [24] propose an adaptive monitoring method (AMM) and a deadline-driven method (DDM) for managing the locations of moving objects for the execution of location-dependent continuous queries. According to the authors, the methods proposed previously focus on the utilization of limited wireless bandwidth, with the issue of providing timely and correct query results to the clients being ignored. The proposed methods generate the location updates as well as maintain the correctness of query evaluation results without increasing the location update overhead.

Ilarri *et al.* [17] present a distributed query processing approach to process continuous location-depended queries by utilizing location granularity in a mobile environment. The authors also focus on the usefulness of appropriate location granularity while processing the query. According to the authors, previously existing approaches did not consider location granularity for the processing of location-depended continuous queries in a distributed and wireless environment. The performance of the proposed query processing approach is analysed in this paper. Results shows that the approach performs well, though the use of location granules create a little more overhead for query processing.

Gupta *et al.* [8] propose a hierarchical database framework for location-depended query

processing in a mobile environment, in order to provide correct and timely query answers to the user with lower communication cost. This framework is based on a tree architecture which works well both for single-zone and multiple-zone queries. The authors also propose an algorithm to support their proposed tree architecture. According to the authors, previously existing approaches for location-dependent continuous query processing focus only on the data management issue and tracking the location of the mobile user. Also, previous approaches are not suitable for multiple-zone queries. The authors analysed the flexibility of their proposed approach and conclude that it is flexible enough for processing both the single-zone and multiple-zone type queries.

Cheng *et al.* [3] propose a time-based location update technique called the query-result update (QRU) for continuous queries over moving objects. The aim of this technique is to control the rate of location updates in order to reduce the query update cost. This technique identifies the objects that have more impact on the query results. Based on this, the QRU calculates the time instant for an object to execute location updates. In addition, the location update frequency of each object is updated continuously to ensure better performance of the techniques. According to the authors, many previously existing update mechanism for continuous queries are based on a distance scheme. There are some existing time-based update mechanisms where an object is assigned a fixed update time interval. The QRU adjusts the time instant assigned to the object dynamically. The authors perform several experiments to evaluate the effectiveness of their proposed techniques and the result shows that the QRU performs well with lower update costs than previously existing methods.

Hong *et al.* [10] propose an efficient Continuous Location Depended Query (CLDQ) processing approach in mobile Wireless Sensor Networks (WSNs) . The aim of this approach is to efficiently process the query and provide correct query results in a fully distributed and self-organized way. They also propose an approach to disseminate the query from the base station to the mobile sensor nodes. In addition, the authors propose a Contention-based Distance-aware Message Scheduling (CDMS) scheme to schedule data

transmission during query propagation and data aggregation. Lastly, they propose an optimization scheme for continuous processing of CLDQs to reduce the query processing cost. According to the authors, the previously proposed approaches are not efficient to process CLDQs in a continuously moving query areas. The authors evaluate the performance of their approach and conclude that the approach performs better in comparison with other existing approaches.

Hsueh *et al.* [11] propose two location update strategies for a trajectory movement model and an arbitrary movement model, respectively to ensure efficient processing of the continuous queries over moving objects. The aim is to reduce the communication cost due to frequent location updates of the objects or the query points. The proposed strategy for a trajectory movement environment is the Adaptive Safe Region (ASR), which utilizes adaptive safe regions to reduce the number of location updates for any data point. For an arbitrary movement environment, the authors propose a Partition-based Lazy Update (PLU) algorithm that utilizes a location information table to make decision for an object to issue a location update. According to the authors, most of the existing work consider an environment where queries are either static or rarely change their locations and are not suitable to reduce the number of unnecessary location updates . The authors perform some experiments to evaluate the performance of their approaches and the outcomes shows that both approaches perform better with low communication cost than existing approaches.

Huang and Lee [15] propose the Possibility-based possible within objects (P2WO) searching algorithm and the Possibility-based possible KNN (P2KNN) searching algorithm to process the continuous range query and the continuous k-nearest neighbor query, respectively for moving objects with uncertain velocity. The authors also propose: (1) a distance-based possibility model to determine the possibility of an object being in the query results, and (2) a pruning strategy with a data partitioning index to prune those objects that are impossible to be in the query result in order reduce the query processing cost. According to the authors, the query processing cost of most of the previous work is very high. In addition,

some of the previous approaches process the spatio-temporal queries by assuming that the velocity of each object is fixed, but in the real world the velocity is not fixed. The authors perform several experiments to evaluate the performance of their proposed approaches and the outcomes show that their approaches perform better than existing approaches in terms of query processing cost. They also conclude that their approaches are suitable for a dynamic environment where the velocity of an object changes very frequently.

Cao *et al.* [2] study about the use and the effectiveness of using the attainability information of an object during Location Depended Query (LDQ) processing in a mobile environment. They present the Reachability Graph that contains the attainability information to index target objects. The authors also present the architecture of a prototype that use the Reachability Graph for LDQ processing. According to the authors, previously existing LDQ processing approaches do not consider the attainability information. They also perform some experiments to evaluate the effectiveness of their strategies.

Lee *et al.* [25] present a query containment technique for Location-Depended Spatial Queries (LDSQs), called LDSQ Containment that makes mobile clients able to determine whether an LDSQ can be answered by using the result of previous queries. The authors also present a system model that supports LDSQ containment. They also formulate containment scope, propose containment scope computation algorithm, containment test algorithm, and optimization techniques for range, window and kNN queries. The aim of their work is to reduce the server workload, query processing latency, and energy consumption of mobile devices and the wireless bandwidth of Location-Based Services (LBSs). The authors propose these approaches to further enhance the performance of existing location-based services. They conduct several experiments to evaluate the performance of their proposed techniques and the outcomes show that their techniques perform better than existing approaches.

Hung *et al.* [16] present a real-time navigation system called Real-Time Traffic Navigation (RETINA) system. It is designed for various Location-Depended Services (LDSs), such as identifying the best path for a mobile client to go from their current position to the

destination. In RETINA, the best path is calculated based on the connecting roads between the starting location and the destination, and considering the current traffic conditions. For their system, the authors formulate the traffic navigation request as Location Depended Continuous Query (LDCQs). The authors propose a replicated dynamic graph approach to manage traffic data needed for searching for the best path. They also propose an adaptive Push or Pull (APoP) scheme for monitoring the best path and the traffic data which improves the performance of their system.

Liu and Hua [28] propose a distributed framework to process moving queries such as moving range query and moving kNN query, over moving objects in a spatial network environment in order to reduce the server loads and location update costs. In this framework, the road is partitioned into road segments and the moving objects only update their location when they move into a new segment. The road segments are used as building blocks for monitoring regions of moving queries in the proposed framework. The authors also propose a concept called Edge Distance, which is used by a moving object to locally compute the network distance without processing a full network map in order to reduce the server workload and save wireless bandwidth. According to the authors, previously existing approaches are designed for an open space environment and cannot be efficiently used in a road network or a spatial environment. The authors also evaluate the performance and conclude that the server workload and communication cost of the query processing can be significantly reduced by their proposed method.

Bordogna *et al.* [1] propose a model for representing and evaluating uncertain location-based queries (LBQ). They classify LBQs into distinct types, depending on the presence of uncertainty affecting the user location, the instance location, and the query scope. The authors propose the evaluation procedure for each type of uncertain LBQ, which consists of two phases: the filter phase, and the refinement phase. The filter phase is used to identify the possible candidate instances satisfying the query. The refinement phase is used to compute the probability of validity of the possible candidate instances. According to the authors,

the previously existing LBQ processing approaches do not consider uncertainty and do not provide the query results with validity estimation.

Iwerks *et al.* [20] propose several approaches to maintain kNN and spatial join queries on continuously moving objects where the information about the moving objects is represented as functions of time. In maintaining queries, these functions are updated by changing the velocity and position of the moving objects as the time advances. The approaches assume no previous knowledge about the updates before they occur. According to the authors, previously existing approaches are not appropriate for efficient maintenance of queries on a continuously moving object.

2.4 Conclusion

In the processing of continuous queries, various research works have been undertaken. In this chapter, we summarize works related to this thesis. In the next chapter, we present our proposed strategies for continuous region query processing.

Chapter 3

Continuous Range Query Processing Strategies

In this chapter, we present two query processing strategies for Location-based Services: one that uses monitoring, and one that does not. The goals of our strategies are: reducing the server workload, reducing the data transmission cost, and lastly reducing the query response time. Both strategies process continuous region queries over moving point sets, where the query issuer object or client is also moving. At some moment in time, the client issues a region query to a central server for point data. The server processes the query and sends a result back to the client only if there is a change in the result from the previous query. For both strategies, we use a spatial indexing [38] mechanism, the mqr-tree [29], to help speed up the searching process.

The remainder of this chapter proceeds as follows. In Section 3.1, we discuss the mqr-tree spatial indexing method, which also includes a very brief description about the mqr-tree region search. In Section 3.2, we describe our proposed continuous region query processing strategy, which does not use monitoring to identify moving points. In Section 3.3, we give an example of our non-monitoring strategy for query processing on the server. Lastly, in Section 3.4, we describe our second continuous region query processing strategy, which is a modification of our proposed first strategy to use monitoring.

3.1 mqr-tree spatial index method

The mqr-tree [29] [34] is a two-dimensional index structure that organizes spatial objects in a two-dimensional node based on different spatial relationships, and helps make the search process and data retrieval more efficient. An object, or subregion of space that contains objects, is represented using a minimum bounding rectangle (MBR), which is the minimum two-dimensional rectangular range that contains objects or subregions. A performance comparison with other strategies shows that the mqr-tree achieves a lower average number of node accesses, which can reduce the query execution time for the region search. In addition, for point data, the mqr-tree has no overlap of subregions and therefore the point search will only proceed along one path and a region search will proceed along significantly fewer paths that may contain no points for the result. We can also modify the region search to also follow only one path. For these advantages, in our strategies, the server uses the mqr-tree spatial access method and a modified mqr-tree region search process in order to perform the query processing task.

3.1.1 mqr-tree structure and objects/points organization within a node of the tree

Each MBR is represented by a set of coordinates, *lx,ly,hx,hx* where *(lx,ly)* is the lower left corner and *(hx,hy)* is the upper right corner of the MBR. In the mqr-tree, a point is represented as an MBR of zero area. That is, the lower corner *(lx,ly)* and the upper corner *(hx,hy)* is the same for a point MBR.

Each node in the mqr-tree [29] [34] is defined by a nodeMBR. For a given node, the nodeMBR encompasses all objectMBRs in the node and all of the nodeMBRs of its descendent subtrees. The mqr-tree node layout is shown in Figure 3.1. Each node has 5 locations: northwest (NW), northeast (NE), southeast (SE), southwest (SW) and center (CTR). Each location of a node contains either:

- (MBR, obj_ptr) where MBR represents a point or an approximation for an object in the database, and obj_ptr is a pointer to it, or

- (nodeMBR, node_ptr) where nodeMBR is the MBR that encompasses all MBRs in the subtree that is pointed to by the node_ptr.

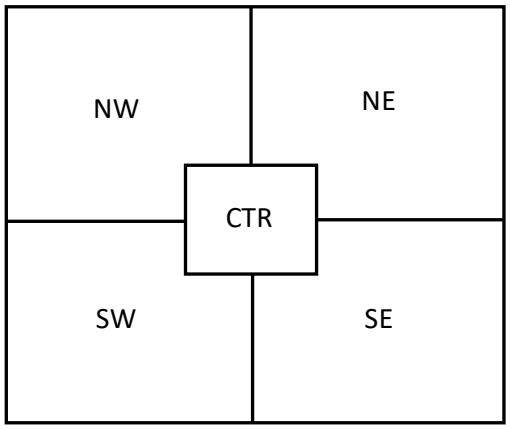


Figure 3.1: mqr-tree Node Layout (from [29])

The appropriate location for each object and each subtree in a node is determined by comparing the center of their MBRs with the center of the node MBR. If the center of the object or subtree MBR is same as the center of the node MBR, then the object or subtree MBR is placed in the CTR location of the node. Similarly, if the center of the object or subtree MBR is at either the northwest, northeast, southeast or southwest position with respect to the center of the node MBR, then the object or subtree is placed in the corresponding location (NW, NE, SE or SW) in the node.

An example of a mqr-tree for a point set is shown in Figure 3.2. In the figure, the root node MBR, Mqr1 which is highlighted in bold lines, represents a region that encompasses the whole set of point MBRs as well as the subtree MBRs. The node for Mqr1 points to two MBRs, Mqr2 and Mqr3 with their associated subtrees. The Mqr2 subtree is northwest and the Mqr3 subtree is southeast of the root node MBR respectively. Therefore Mqr2 and Mqr3 are located in the NW and SE locations respectively. Mqr2 encompasses all of the MBRs within the northwest region. Mqr2 references a subtree Mqr6 at the southwest location and a point P1 at the northeast location. Similarly, Mqr3 encompasses all of the MBRs within the southeast region. Mqr3 also references two other subtrees, Mqr4 and Mqr5. The Mqr4 MBR and Mqr5 MBR encompasses all of the points within their respective regions.

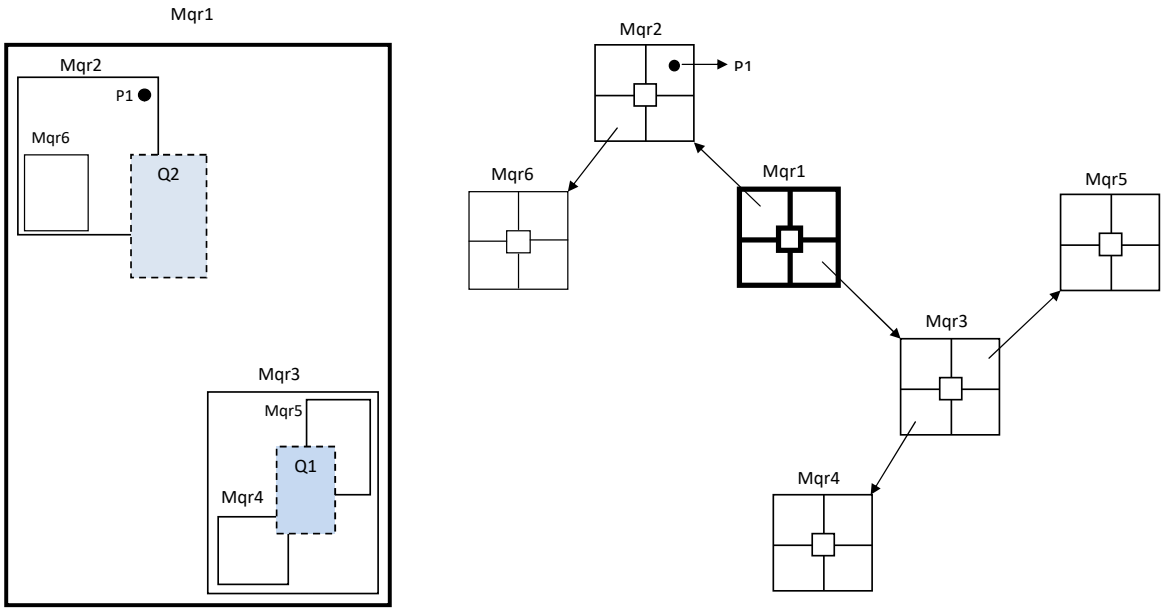


Figure 3.2: An example mqr-tree and superMBR for a query

3.1.2 mqr-tree Insertion

In the mqr-tree, when inserting a new object, at first the root nodeMBR is adjusted to include the new object [34]. Then the appropriate location (NE, SE, SW, NW, or CTR) is identified for inserting the object. If the identified location is empty, the object is inserted. If it contains another object, a new child node is created for that location, and both the new and existing object are placed in the new node. If the location contains a subtree pointer, then the insertion location search process continues in the same way in the subtree.

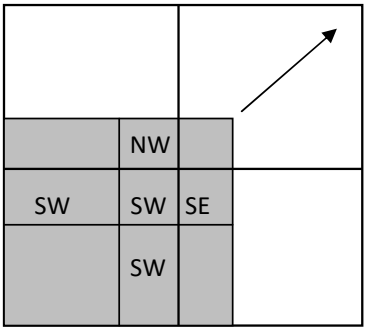


Figure 3.3: Object displacement due to node MBR extension [34]

During the insertion process, a change in the node MBR may result in changes to the other existing object or subtree orientations with respect to the center of that node MBR.

Figure 3.3 shows an example where the node MBR has been extended (the outer rectangle) to insert a new object, so that the object can fit in that MBR. The gray rectangle in the figure represents the former node MBR. We can see that, parts of the NE location of the former node MBR has become part of the NW, SW, SE and CTR (not shown) locations of the extended MBR respectively. Similarly, other locations of the former node MBR have changed or displaced for the extension. For this reason, all MBRs (i.e., object or subtree) whose centers are in these locations, are removed and re-inserted or relocated, in the same manner as a new object is inserted. The insertion process for the displaced objects start from their current node. This process will continue until either: (1) no more node MBR changes occur, or (2) changes in any node MBRs do not displace the other MBRs or subtrees.

3.1.3 mqr-tree Region Search

In the mqr-tree region search process [34], the search for a region query is started from the root node. The query region is tested in order to find overlap with any existing MBR in each of the five locations of the root. If a location is empty, then the search does not continue via that location. If a location contains a MBR that overlaps with the query region, then the search process continues in one of the following ways:

- If the location contains an object, then the corresponding object MBR is returned from the database as part of the result.
- If the location contains a subtree pointer, then the same search repeats in the corresponding subtree.

3.2 Our Original (first) Continuous Region Query Processing Strategy

In this section, we describe our first strategy which we call our Original Strategy. In Section 3.2.1, we describe the overall procedure of our Original Strategy. The next three sections describe the strategy in detail. Section 3.2.2 deccribes how the superMBR for a query is identified in the Original Strategy. Section 3.2.3 presents the process of fetching

all the points within the superMBR region. In Section 3.2.4, we describe how the server in the Original Strategy compares the point set.

3.2.1 Overview

The client issues a query via a wireless network. The server receives the query, which starts the search process. For the first query from a client, the server locates a superMBR (defined in Section 3.2.2) and then fetches all of the points within the superMBR. The server then sends the superMBR and the resulting point set to the client. When this client issues the second query, the server locates a new superMBR and corresponding point set for this query. In this strategy, no monitoring is used to identify points that have moved before the next query arrives. The server only compares the new superMBR with the previous one to identify changes in the new result set in the following way:

- If the superMBRs are different, the server then sends the new superMBR and the point set to the client.
- If the superMBRs are the same, then the server must compare the new point set with the previous point set in order to identify the differences between them.
  - If the point sets are different, then the server sends the new point set and superMBR to the client.
  - If the point sets are the same, then the server only sends a message indicating that the new result is the same as the previous one, instead of sending the complete point set with its superMBR.

3.2.2 Locating the superMBR for a query

A superMBR (Super Minimum Bounding Rectangle) is the last nodeMBR along the query path where the query region is fully enclosed. This is illustrated with an example. In the Figure 3.2 (page 37), suppose, a client issues a region query Q1 (a shaded rectangle).

For this query, the Mqr3 MBR is the superMBR because this is the last node MBR along the query path in the tree that wholly contains the query region Q1. Also note that Mqr4 overlaps the query region Q1 but does not fully contain it and so it can not be the superMBR for Q1.

According to our strategy, the server sends the identified superMBR (in this case, Mqr3) and all of the point MBRs that exist within this super region, to the client (if required) in response to the region query Q1. If we consider another region query Q2 in Figure 3.2 (page 37), the superMBR is the root MBR, which is Mqr1. This is because only Mqr1 fully contains the query region Q2. So for query Q2, the server will return Mqr1 as the superMBR and all of the points contained in the mqr-tree, to the client (if required).

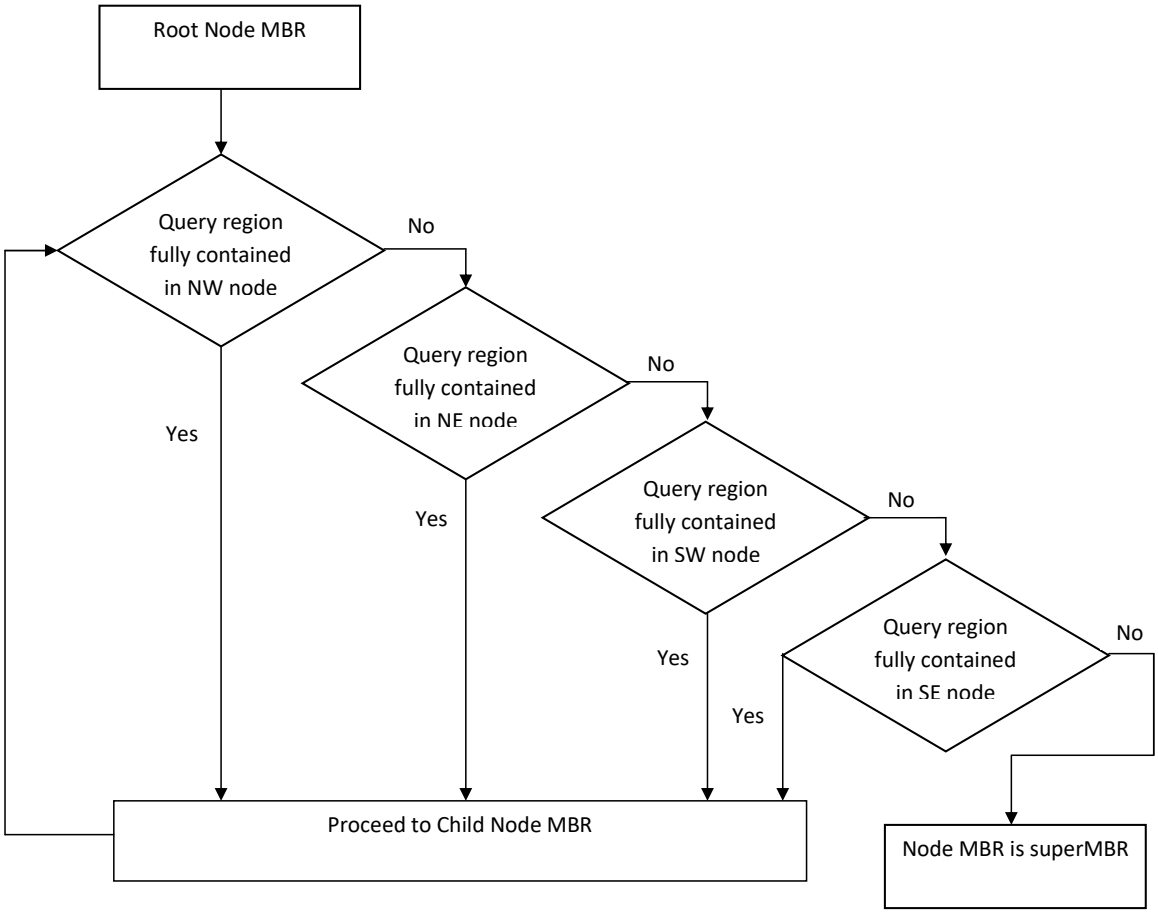


Figure 3.4: superMBR Identification Process

Figure 3.4 shows the process for identifying the superMBR for a query region. The

server receives the region query from the client, and starts the search process to find the node with the superMBR for the query. If the query region is fully contained in the root node, then it is tested to see if it is fully contained in one of the NW, NE, SW and SE child nodes of the root node. If the query region is not fully contained in any of the four child nodes (NW, NE, SW and SE) of the root node, then the search process identifies the MBR of the root node as the superMBR. If the query region is fully contained in the MBR of one of the child nodes of the root node, then the search process continues in the chosen subtee. This process is repeated until the region query is not fully contained in any child node MBRs.

3.2.3 Point set generation within the superMBR

The superMBR defines a region that encompasses the query region, all subtreeMBRs within the query region, as well as all other existing pointMBRs within the super region. The point set is gathered by traversing the subtree pointed to by the superMBR node. An example is shown in Figure 3.5.

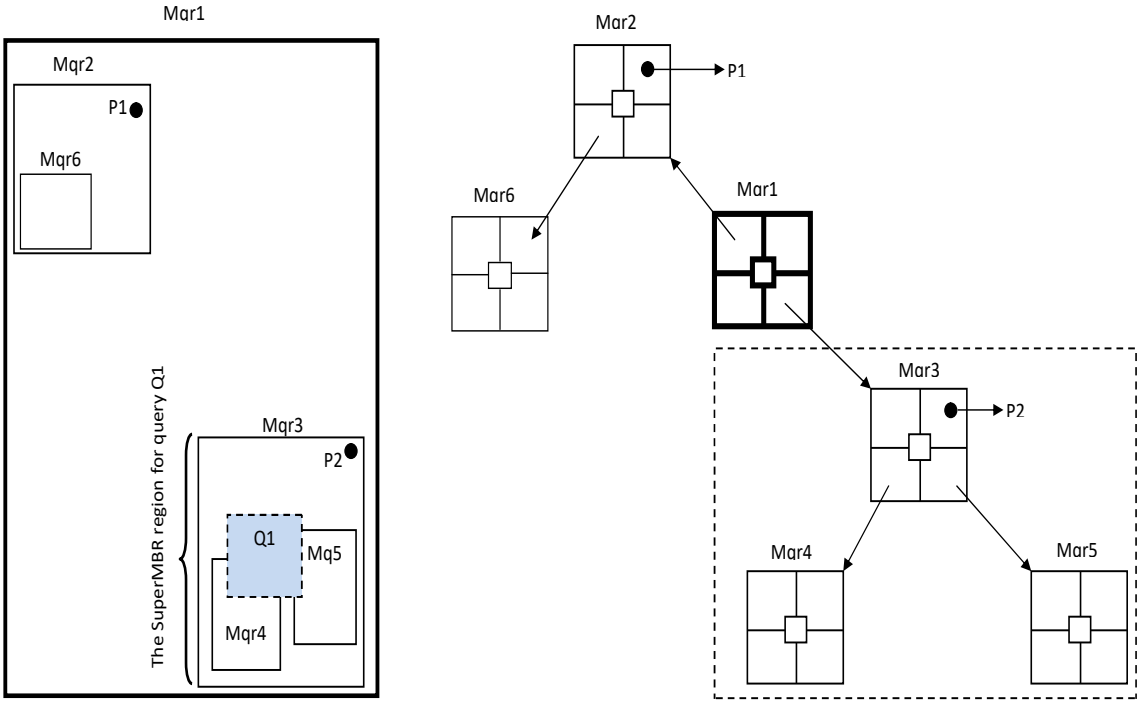


Figure 3.5: Point set generation within the superMBR

Here the Mqr3 MBR is the superMBR for region query Q1. In the figure, the three nodes inside the dashed rectangle are within the superMBR region. The server fetches all of the pointMBRs that are contained within these three nodes.

3.2.4 Comparing the New and Old superMBR and Point Set

After processing a region query, the server saves the results. When the server receives and processes the next query, the server compares the new result with the previous result. The overall comparison process is shown in Figure 3.6.

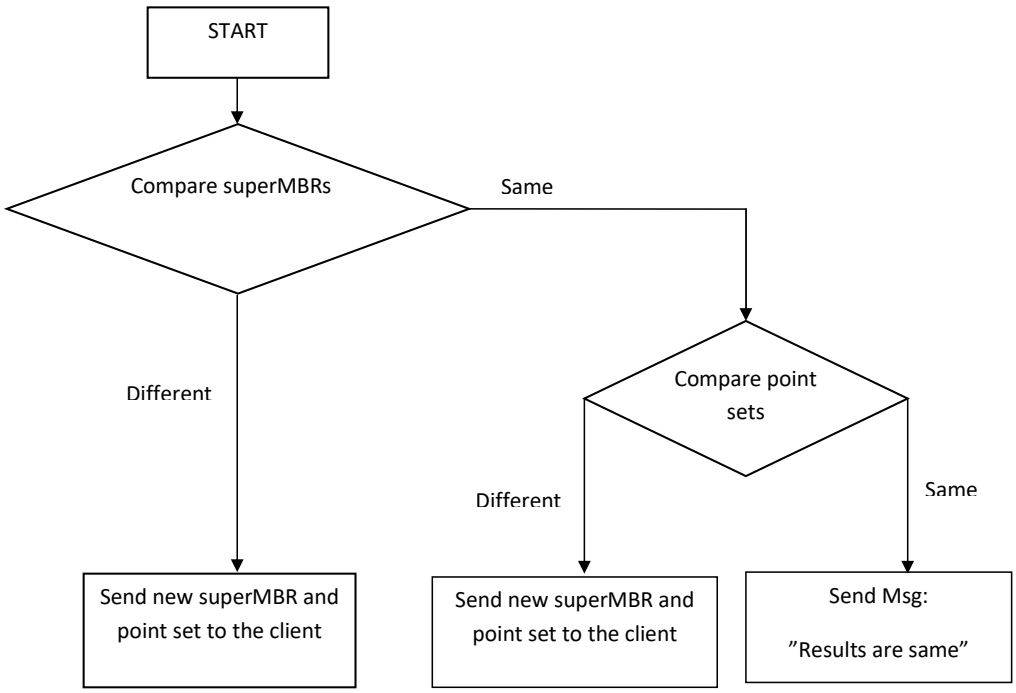


Figure 3.6: Comparing the new and old superMBR and point set

In this process, one of two things can happen: only the old and new superMBRs need to be compared, or the new and old superMBRs as well as the point sets need to be compared.

- First, compare the new superMBR with the previous superMBR:
  - If the superMBRs are different, the new data set and superMBR are sent to the client

  - If the superMBRs are the same, then the point sets need to be compared

    This is because although the superMBRs are same, the residing point set can be different. For example, when some point moves within or outside of the region identified by the superMBR but the points on the border region remain unchanged, then the superMBR remains unchanged but the point set changes.

- If the new point set needs to be compared with the previous point set:

  - if the point sets are different (e.g. there is at least one point that has moved), then the new data set and superMBR are sent to the client.
  - if the point sets are the same, a message indicating that *The result is same as the previous result* is sent to the client, instead of sending the entire result file.

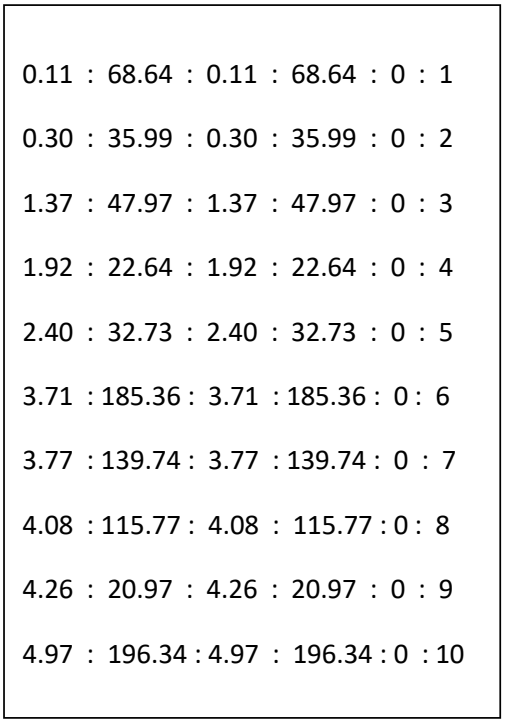


Figure 3.7: A sample point data set

3.3 Example

Here an example is given that illustrates the proposed Original Strategy. A sample point data set is shown in Figure 3.7.

Each point is represented here as:

$$lx : ly : hx : hy : 0 : ID$$

where *(lx,ly)* is the lower left corner and *(hx,hy)* is the upper right corner of the MBR for a data. Since we are working with point data, the lower and upper corners are the same for each point MBR.

3.3.1 mqr-tree for the Sample Data

The mqr-tree for the sample data is shown in Figure 3.8. Here Mqr1 is the root node which points to two nodeMBRs with their associated subtree: Mqr2 in its NE region and Mqr3 in its SW region. Mqr1 also references a point P9 in its SE location. Mqr2 references a nodeMBR and subtree Mqr4 in its SW region. It also contains two points: P6 and P10 in the NW and NE locations respectively. Mqr3 points to a subtree (and nodeMBR) Mqr5 in its SE region. It also references three points P1, P3 and P2 in the NW, NE and SW locations respectively. Mqr4 references two points: P7 in the NW and P8 in the SE locations. Mqr5 references two points P5 and P4 in the NE and SW locations respectively.

The coordinate *(lx ly hx hy)* of each node MBR and points are as follows:

Mqr1 $\rightarrow$ 0.11 20.97 4.97 196.34

Mqr2 $\rightarrow$ 3.71 115.77 4.97 196.34

Mqr3 $\rightarrow$ 0.11 22.64 2.40 68.64

Mqr4 $\rightarrow$ 3.77 115.77 4.08 139.74

Mqr5 $\rightarrow$ 1.92 22.64 2.40 32.73

P1 $\rightarrow$ 0.11 68.64 0.11 68.64

P2 $\rightarrow$ 0.30 35.99 0.30 35.99

P3 $\rightarrow$ 1.37 47.97 1.37 47.97

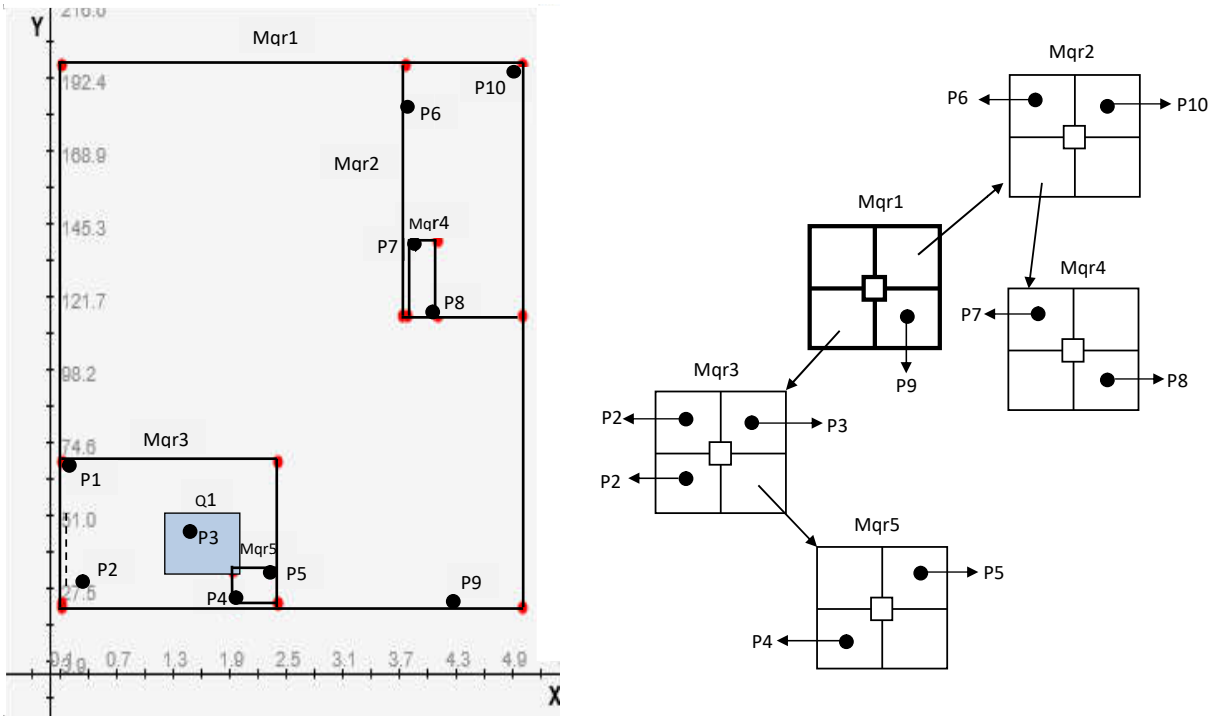


Figure 3.8: The mqr-tree and a query Q1 for the sample data in Figure 3.7

P4 → 1.92 22.64 1.92 22.64

P5 → 2.40 32.73 2.40 32.73

P6 → 3.71 185.36 3.71 185.36

P7 → 3.77 139.74 3.77 139.74

P8 → 4.08 115.77 4.08 115.77

P9 → 4.26 20.97 4.26 20.97

P10 → 4.97 196.34 4.97 196.34

3.3.2 Query Processing

A client sends a region query, suppose Q1 (as shown in the shaded rectangle in Figure 3.8) to the server. The coordinate *(lx ly hx hy)* of Q1 is:

Q1 → 1.12 34.56 1.95 51.46

After receiving the query Q1, the server starts the search process. Starting from the root

node Mqr1, it checks to see whether Q1 exists in the NW, NE, SW or SE region of Mqr1. It is seen that Q1 exists in the SW region of Mqr1 as in Figure 3.8. So, the search process proceeds along the subtree Mqr3. It is found that Mqr3 is the last node which fully contains Q1. So, Mqr3 is identified as the superMBR for Q1 (described in Section 3.2.2). Now the server traverses the subtree Mqr3 to generate the set of points that are residing within the superMBR region (described in Section 3.2.3). The server fetches all of the pointMBRs within Mqr3, as well as fetches all of the pointMBRs within Mqr5, which is a subtree also pointed to by the Mqr3. The server now sends the superMBR Mqr3 and the generated point set to the client as the result of query Q1. The result of the region query Q1 is shown in Figure 3.9.

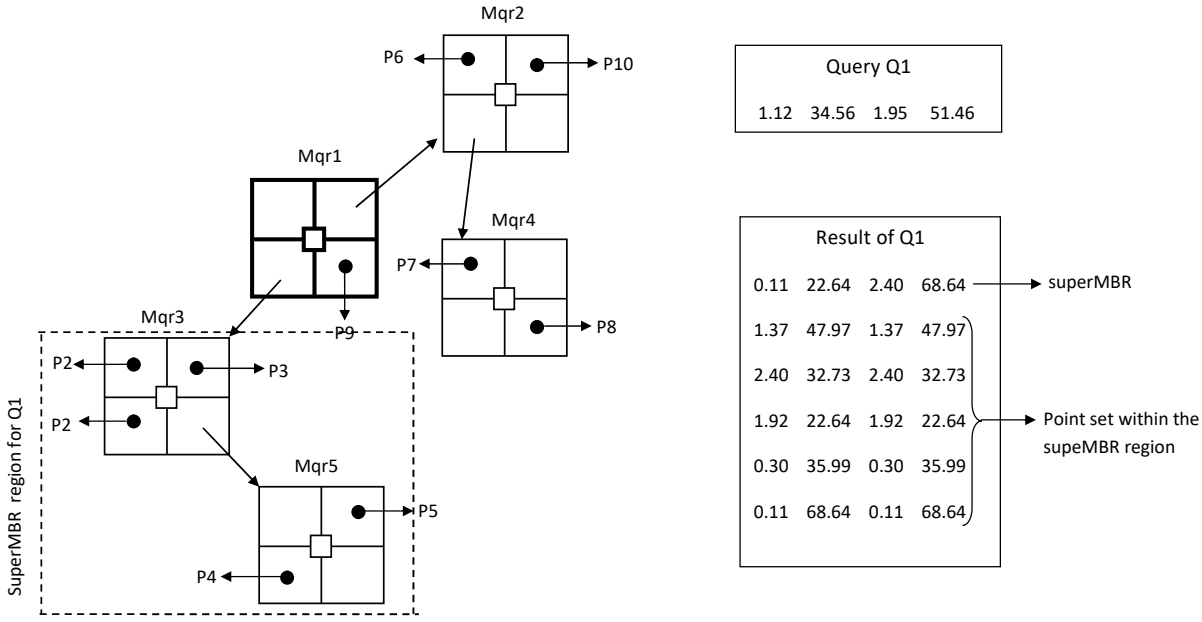


Figure 3.9: Result of the Region Query Q1

The client sends a second query Q2 as shown in Figure 3.10. The coordinate *(lx ly hx hy)* of Q2 is: (0.45, 39.61, 1.94, 64.22). The server then identifies Mqr3 as the superMBR for Q2, and fetches all of the pointMBRs within this Mqr3 region. The server now compares the new superMBR (i.e., Mqr3) with the previous superMBR (described in Section 3.2.4) which is the superMBR for Q1, and finds that they are same. So, the server now compares the new point set with the previous point set of Q1. It is found that the point sets are also

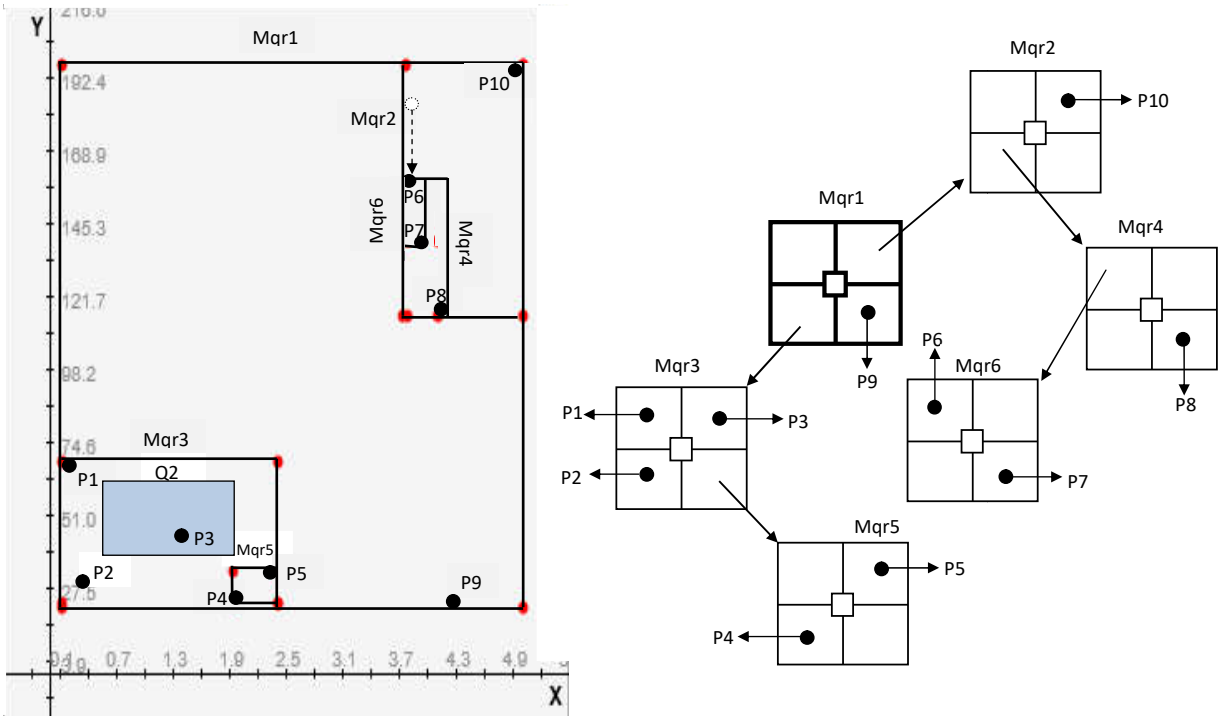


Figure 3.10: The mqr-tree and the second query Q2

the same. In Figure 3.10, we can see that the point P6 has moved (as shown by a dashed arrow) to a new location of coordinate (3.71, 153.36, 3.71, 153.36). This change of P6's location also changes the tree structure which we can see in Figure 3.10 (i.e., Mqr4 is splits into two: Mqr4, and Mqr6). Since the point P6 resides and moves outside of the superMBR region, it does not affect the new result set. The superMBR and the point set of Q2 is the same as the previous result, so the server only sends a message indicating the result is same as before, instead of sending the entire result file.

The client sends a third query Q3 identified by (1.30, 35.99, 1.97, 63.2) is shown in Figure 3.11. The server again identify the Mqr3 as the superMBR of Q3, and fetches all the pointMBRs within this region. After finding the same superMBR, the server compares the new point set with the point set of the previous query Q2. The server finds that the new point set is different. From Figure 3.11, we see that point P2 has moved to a new location (0.60, 35.99, 0.60, 35.99), and also that point P6 has moved back to its first location. The point P2 is inside the superMBR region and has changed the result set. So, the server sends

the new point set with the superMBR to the client.

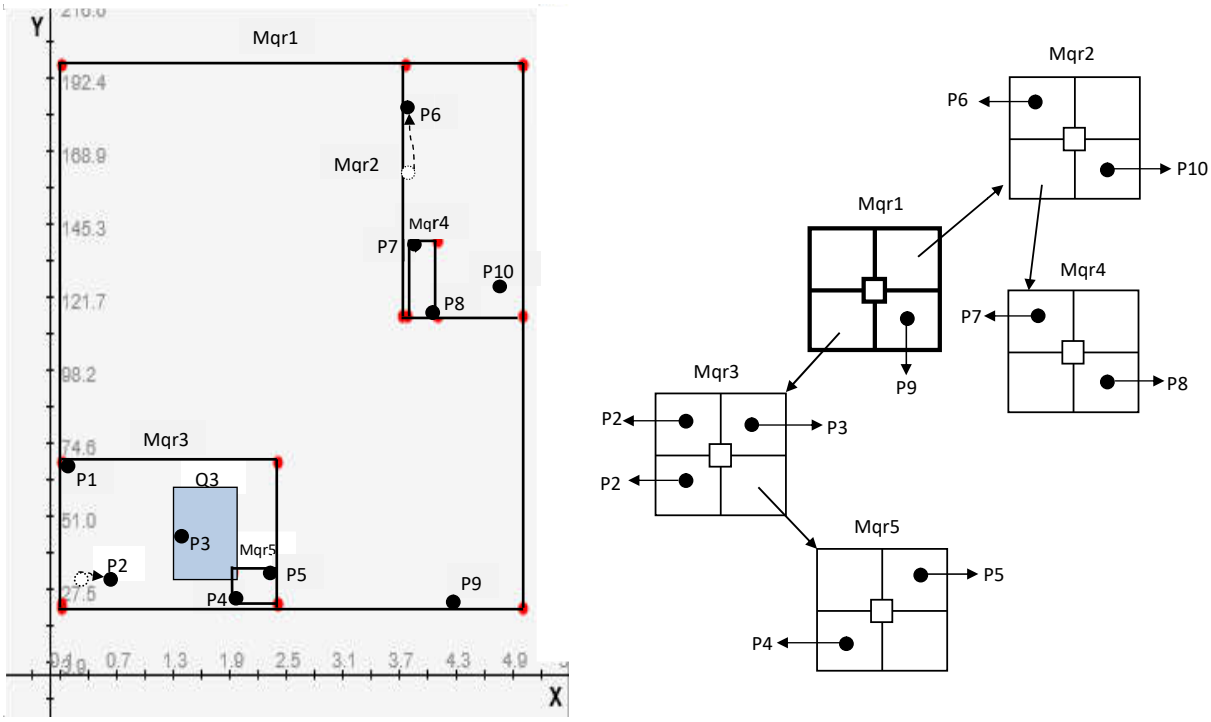


Figure 3.11: The mqr-tree and the third query Q3

Figure 3.12 shows a fourth query Q4 identified by (1.98, 22.00, 2.37, 31.80), sent by the client. The server identifies Mqr5 as the superMBR for Q4 (because Q4 is fully contained in the Mqr5 region), and then fetches all the pointMBRs within Mqr5. The server now compares the new superMBR with the superMBR of previous query Q3 and finds that they are different. So the server sends the new point set with its superMBR to the client without comparing the new point set with the point set of Q3.

3.4 Improved Strategy for Continuous Region Query Processing

A SuperMBR can be the same for different queries. In spite of the same superMBRs, the point sets can be different as we mentioned in 3.2.4. For this reason, the old and new point sets must be compared when the superMBRs are the same, as we have done in our first strategy. When the superMBR for a query identifies a large region, then the residing point set can be large too. For example, sometimes the rootMBR can be the superMBR for

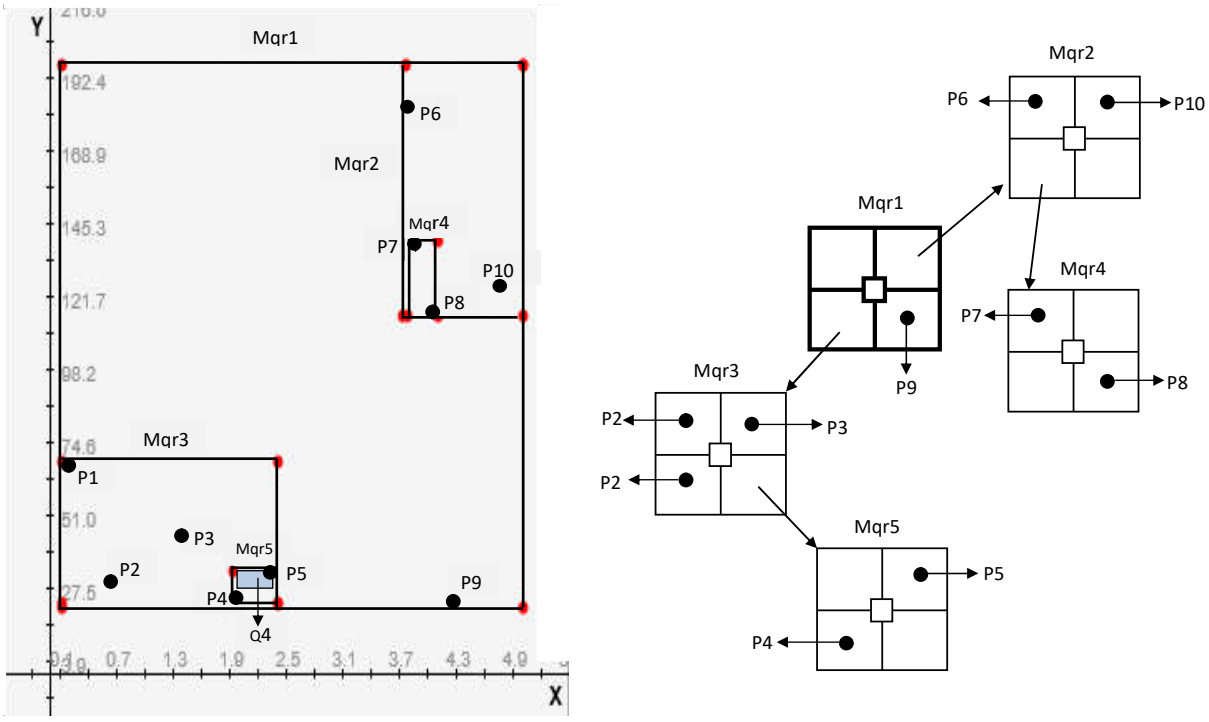


Figure 3.12: The mqr-tree and the forth query Q4

a query as we can see for Q2 in Figure 3.2 (page 37), and then the whole data set becomes the resulting point set for the query. As a consequence, comparing all points between two large point sets results in significant overhead for the server. By considering this issue, we propose another strategy called our Improved Strategy, in order to reduce the overhead of comparing large point sets. This strategy uses monitoring to track any points that have moved between the transmission of two query regions. The server also keeps a record of the old points that have moved in the new locations.

3.4.1 Overview

After receiving a query from a client, the server starts the search process. At first, the server identifies the superMBR for the query and then fetches all of the points within the superMBR, following the same procedure as our original strategy described in Sections 3.2.2 and 3.2.3. The server sends this superMBR and the point set to the client. When the same client sends a second query, the server again proceeds with the search process to

identify a new superMBR and residing point set for the new query. In addition to this, in our improved strategy the server monitors the moving points. With a moving point data set, following three situations may occur:

- point(s) may moves anywhere within the rootMBR region.
- point(s) may be inserted anywhere inside the rootMBR region.
- point(s) may moves from anywhere and go outside of the rootMBR region.

Thus the resulting point set for the consecutive query can be changed depending on the above situations. In this strategy, the server keeps track of the changed new points as well as the old location of the points that have moved.

After generating the result for the next query from the client, the server compares the new superMBR with the previous one:

- If the superMBRs are different, the new data set and superMBR is sent to the client.
- If the superMBRs are the same, then the server monitors the location of any moving point(s) with respect to the superMBR:
  - If any moved point (which is new to the current data set) resides in the superMBR, then the new point set and superMBR are sent to the client.
  - If a moved point does not reside in the superMBR, the server checks the old points (which have been moved to a new location) to see whether any of these points resided in the superMBR:
    - If any old point (which has been moved to a new location) resided in the superMBR, the server sends the new data set and its superMBR to the client.
    - If no old point (which has been moved to a new location) resided in the superMBR, then the server only sends a message to the client indicating that the result is same as the previous one, instead of sending the whole result set.

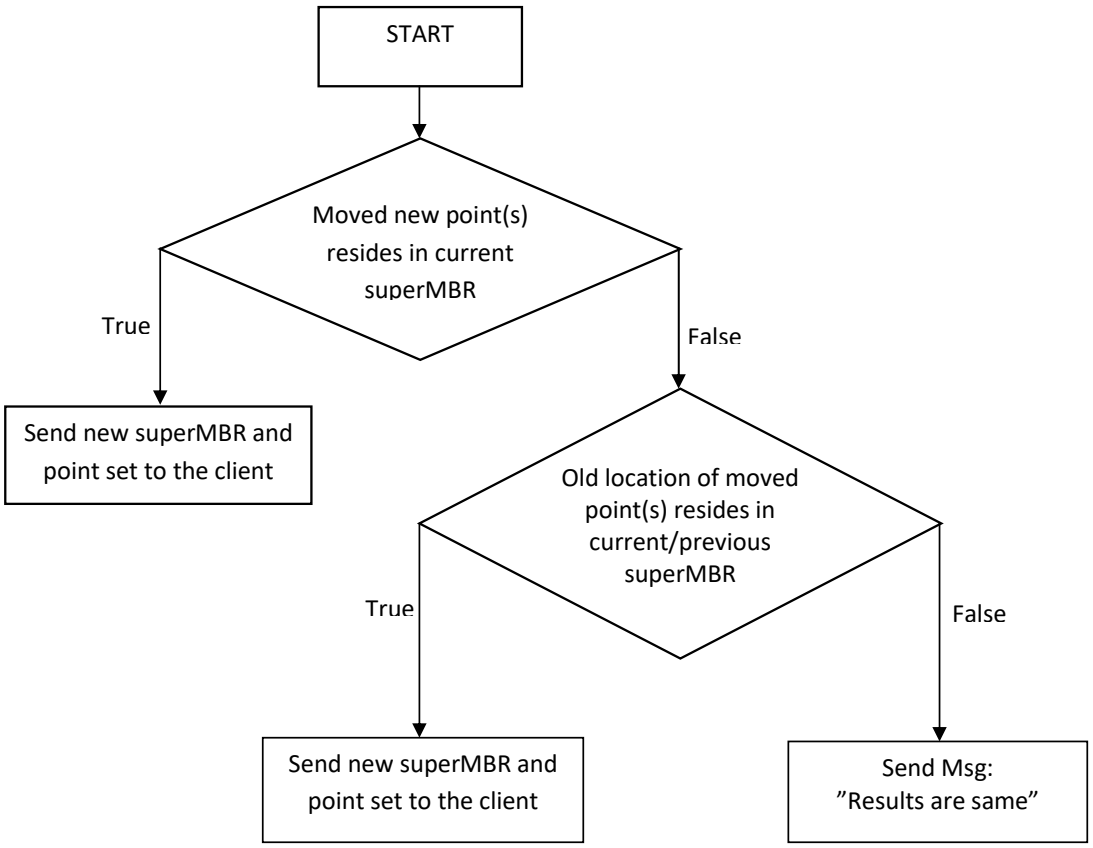


Figure 3.13: Test the moved point(s) with respect to the superMBR

3.4.2 Monitoring the Moving Point(s) with respect to the superMBR

Figure 3.13 shows the process of testing any point that has moved. The server processes the query as well as monitors and keeps a record of the moving points within the data set. After comparing the new and old superMBR, when the server finds that the superMBRs are the same, then it tests the moved new point(s) in the dataset to identify whether it resides within the area identified by the current superMBR. If any one of the moved new points resides in the superMBR region, the server sends the superMBR and the new point set to the client. If no moved point resides in the superMBR region, the server tests the old location of the moved point(s) to see whether they resided in the superMBR region. If any of the old points that have moved reside in the superMBR region, it indicates that the old points were located previously in the superMBR region but currently have moved. So there is a change in the current point set and the server sends the superMBR and the new point

set to the client. If the old points do not reside in the superMBR region, it means that there is no change in the new point set. The server sends only the message to the client indicating that the result is same as before.

3.4.3 Example

We consider the example scenarios in Section 3.3 over the data set in Figure 3.7 (page 43) for the improved strategy. For a first query Q1 in Figure 3.8 (page 45), the server with our improved strategy identifies Mqr3 as the superMBR and generates the residing point set by using the same procedure as the original strategy. The server then sends the whole result file to the respective client.

For a second query Q2 in Figure 3.10 (page 47), the server identifies Mqr3 as the superMBR and fetches all the residing points. The server also monitors the moving points and finds that the point P6 has moved to a new location. The server keeps a record of the previous location and new location of P6 in two files as follows:

- P6(new) $\rightarrow$ (3.71, 153.36, 3.71, 153.36)
- P6(previous) $\rightarrow$ (3.17, 185.36, 3.17, 185.36)

Meanwhile, by comparing the superMBRs, the server finds that the new superMBR is the same as the previous. Instead of checking all of the points within the superMBR as in the original strategy, the improved strategy tests only *P6(new)* to see whether it resides in the current superMBR region. It is found that *P6(new)* does not reside in the Mqr3 region. Now, the server tests the *P6(previous)* and finds that it is also not in the superMBR region. That means there is no change in the result set and the server sends the message indicating that the result is the same as previous.

For the third query Q3 in Figure 3.11 (page 48), the server identifies Mqr3 as the superMBR and generate all the points within it. The server monitors that P2 and P6 have moved and keeps their previous and new location records. Since the previous and new superMBRs are the same, the server tests *P2(new)* with respect to the current superMBR

Mqr3 and finds that *P2(new)* resides in this region. Thus, there is a change in the new result set and the server sends the entire point set with its superMBR to the client.

In case of the fourth query Q4 in Figure 3.12 (page 49), our improved strategy does the same thing as the original strategy. This is because after generating the result, the server finds that the new and previous superMBRs are different and so it sends the entire result file to the client.

Now, consider a fifth query Q5 as shown in Figure 3.14. For this query the server identifies Mqr3 as the superMBR and generates the residing point set. The server monitors that P3 has moved as shown by the dashed arrow in Figure 3.14. Since the new superMBR is same as the previous, the server tests the *P3(new)* and found that it does not reside in the Mqr3 region. Now the server tests the *P3(previous)* and finds that it was in the Mqr3 region. That means that a point from the previous result has moved outside of the superMBR region. So there is a change in the new result set and the server sends the entire result file to the client.

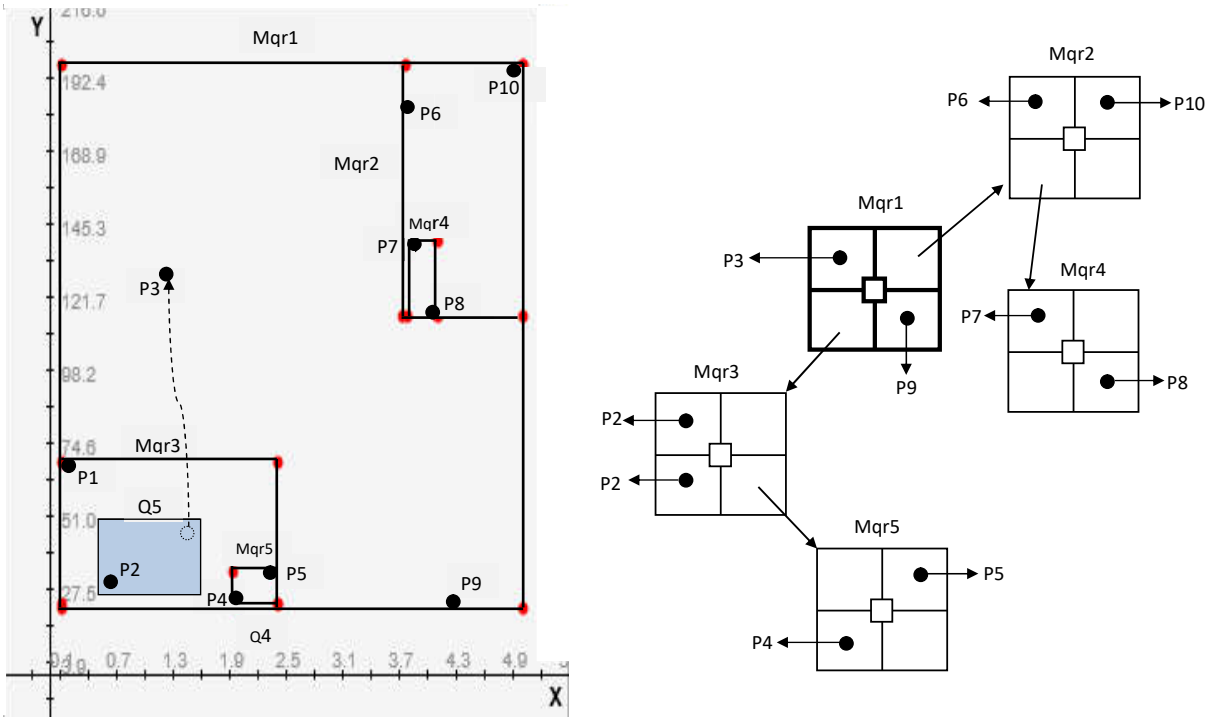


Figure 3.14: The mqr-tree and the fifth query Q5

3.5 Conclusion

This chapter presents our proposed two strategies for continuous region query processing of location-based services. The next chapter presents the performance evaluation of these strategies.

Chapter 4

Experiments and Evaluations

In this chapter, we evaluate the performance of our proposed query processing strategies through experiments. The objectives of our method are to: (1) reduce the server workload, (2) reduce the data transmission cost and (3) reduce the query response time, for location-based services while providing an answer for a continuous region query. We first compare our original strategy with a brute-force (BF) method. We found that our strategy can significantly reduce the server workload and data transmission cost over the brute-force method. We propose an improved strategy in order to minimize the query response time. The data transmission cost and server workload of our improved strategy are same as the original strategy. We also compare our improved strategy with the original strategy and brute-force strategy. The experimental results show that the improved strategy achieves lower query response times than the original and brute-force strategy.

This chapter is organized as follows: In Section 4.1, we briefly describe the brute-force strategy. In Section 4.2, we describe the settings of our experiments. We also describe the data sets we use in our experiment scenarios. The performance metrics used in our evaluations are discussed in Section 4.3. In Section 4.4, 4.5, and 4.6, we present the performance evaluation of our strategies for different scenarios.

4.1 Brute-force strategy implementation

In the brute-force (BF) strategy for continuous region query processing, every time a client issues a query, the server process the query and always sends the entire result back to

the client. For our experiments, we implement the BF strategy as follows:

- The client issues query via a wireless network.
- The server receives the query and starts the search process in order to locate the superMBR.
- Then, the server fetches all of the points within the superMBR.
- The server then sends the superMBR and the resulting point set to the client.

Unlike our proposed strategies in Section 3.2 and 3.4, the BF strategy sends the entire file of results to the client for every query. It does not compare the new query result with the previous result to see whether they are the same or not.

4.2 Experiment setup

We simulate a typical client server application (e.g., Location-based service) for our experiments. In our environment, a moving client issues the queries to a central server. The server processes these spatial queries and sends the answer to the client.

We evaluated our query processing strategies in different scenarios:

- Moving client issues queries on static data.
- Moving client issues queries on moving data.
- Moving client issues queries on moving data, where the number of moving points on the data set varies.

We implemented the server side processing using the C programming language and the client using Java for Android programming. All the experiments were carried out on an Intel(R) Core(TM) i7-5500U CPU@2.40 GHz with 8 GB RAM computer, which is powered by the Windows 10 Home (Version 1703) operating system.

4.2.1 Static data set

For our experiments, we use six synthetic data sets that contain 500, 1000, 5000, 10000, 50000, and 100000 points respectively. As we show in Figure 3.7 (page 43), each point in the data set is represented as: *lx : ly : hx : hy : 0 : ID*, where *(lx,ly)* is the lower left corner and *(hx,hy)* is the upper right corner of a point MBR.

Given a data file containing X points, these points are randomly generated over an $(\sqrt{X} \times \sqrt{X})$ area. An example in Figure 4.1 shows 16 points over an (4 x 4) area.

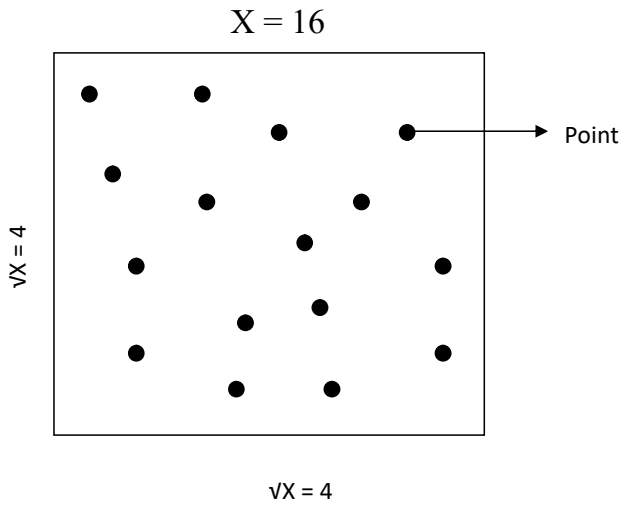


Figure 4.1: 16 points over an (4 x 4) area

4.2.2 Moving data set

Here, we also use six synthetic data sets. Each data set contains X points (i.e., in our case 500, 1000, 5000, 10000, 50000, 100000 points for the six data sets respectively). For each X, ten transition files were created, each moving a point from the previous file. For example, in the 500 points data set of Figure 4.2, the *up_500-0.fnl* contains the starting point set. The *up_500-1.fnl* moves a point from the *up_500-0.fnl*, the *up_500-2.fnl* moves a point from the file *up_500-1.fnl* and so on. For the ten files of each data set, the 1st, 3rd, 5th, 7th and 9th file have no mqr-tree structural change, and the 2nd, 4th, 6th, 8th file have mqr-tree structural change.

Figure 4.2: 10 files for a data set of 500 points

4.2.3 Varying moving points in the 1000 points data set

For the experiments of the third scenario, we generated five 1000-point data sets with different numbers of moving points (i.e., in our case 1, 2, 3, 4 and 5). Again, we generate ten transition files for each data set (i.e., *up_1000-0.fnl, up_1000-1.fnl,.....,up_1000-9.fnl*), in the same manner as described in Subsection 4.2.2.

In the first data set, one point moves per transition, while in the second data set two points move per transition and so on. Here, also for the ten files of each data set, the 1st, 3rd, 5th, 7th and 9th file have no mqr-tree structural change, and the 2nd, 4th, 6th, 8th file have mqr-tree structural change.

4.2.4 Region queries

For each data set X (where X is the number points in a data set), ten region queries of size (10x10) are created, that traverse along the middle of the ($\sqrt{X} \times \sqrt{X}$) area. Each query moves ten units along the diagonal. These queries are applied to both the static and moving point data sets.

4.3 Performance Metrics

We compare our original strategy, improved strategy and the brute-force strategy in terms of three performance metrics:

- Data transmission cost: This is the cost of transmitting the result of each query to the client, by the server.

  We calculate this cost by considering:

  - 8 bytes per floating point number
  - 4 bytes per integer
  - 1 byte per character

- Result-sending workload: This is the number of times the entire result set has to be sent to the client.

  As we described in 3.2 (page 38), for the first query the server sends the entire result set to the client. For the next query, the server compares the new result set with the previous one and decides whether the entire result set should be sent or not. This performance metric compares the number of times the entire result set is sent to the client over the entire trajectory.

- Query response time: The response time measured in microseconds, shown in Figure 4.3.

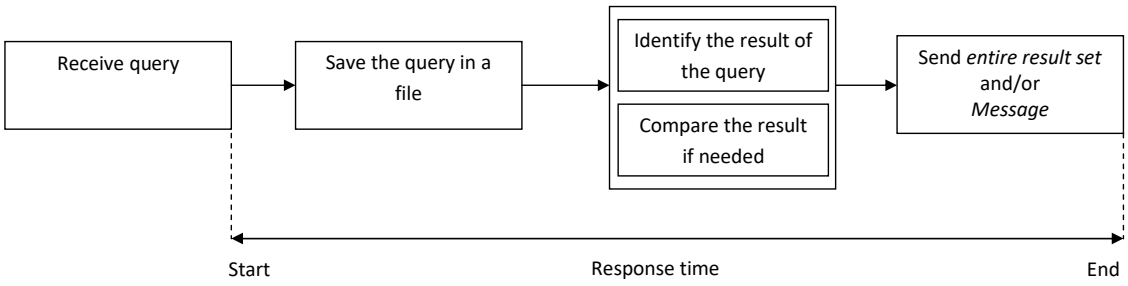


Figure 4.3: The response time

We measure the time from receiving the query at the server until identifying and sending the result back to the client.

The improved strategy has the same data transmission cost and result-sending workload as our original strategy. So, only the original strategy and BF strategy are compared in terms of these two performance metrics. We compare the improved strategy, the original strategy, and BF strategy in terms of the query response time.

4.4 Scenario 1: Moving client issues query on static data

In this test, the client issues ten region queries on the static data sets (as described in Section 4.2). We evaluate the performance of our strategies in terms of the performance metrics that are discussed in Section 4.3. We compare our original strategy and the BF strategy with respect to the data transmission cost and result sending workload. We compare the response time of our improved strategy with both the original and BF strategy.

4.4.1 Data transmission cost results

Figure 4.4 shows the comparison of our original strategy and the BF strategy in terms of the average data transmission cost over 10 range queries for the six data sets. The result shows that our strategy has a lower average data transmission cost for all data sets. This is because in our strategy, the server does not send the entire result file when the current results are the same as the previous results, but the BF strategy does. The percentage (%) of the average data transmission cost improvement of our original strategy over the BF strategy is also shown in the figure.

From Figure 4.4 we can see that for the 1000, 10000 and 50000 points data sets, the transmission cost is almost double for the BF strategy. One reason for this is the following. There may be a situation when the query search process identifies the rootMBR as a superMBR. Then the server needs to send the superMBR and all the points in the data set to the client as mentioned in the example in Section 3.2.2. For the 1000, 10000 and

50000 points data sets, the search process identifies the rootMBRs as the superMBRs for two consecutive queries from the client. For these cases, our strategy sends the entire result file only once, but the BF strategy sends it twice.

Data sets	Data transmission cost		
	Original Strategy (OS)	Brute-force Strategy (BF)	BF/OS (%)
500	1932.4	2300.8	16.01182
1000	3801.2	7318.4	48.05969
5000	16975.6	17442.2	2.675121
10000	32477.3	64764.8	49.85347
50000	160428.4	320761.6	49.98516
100000	320489.2	320825.6	0.104854

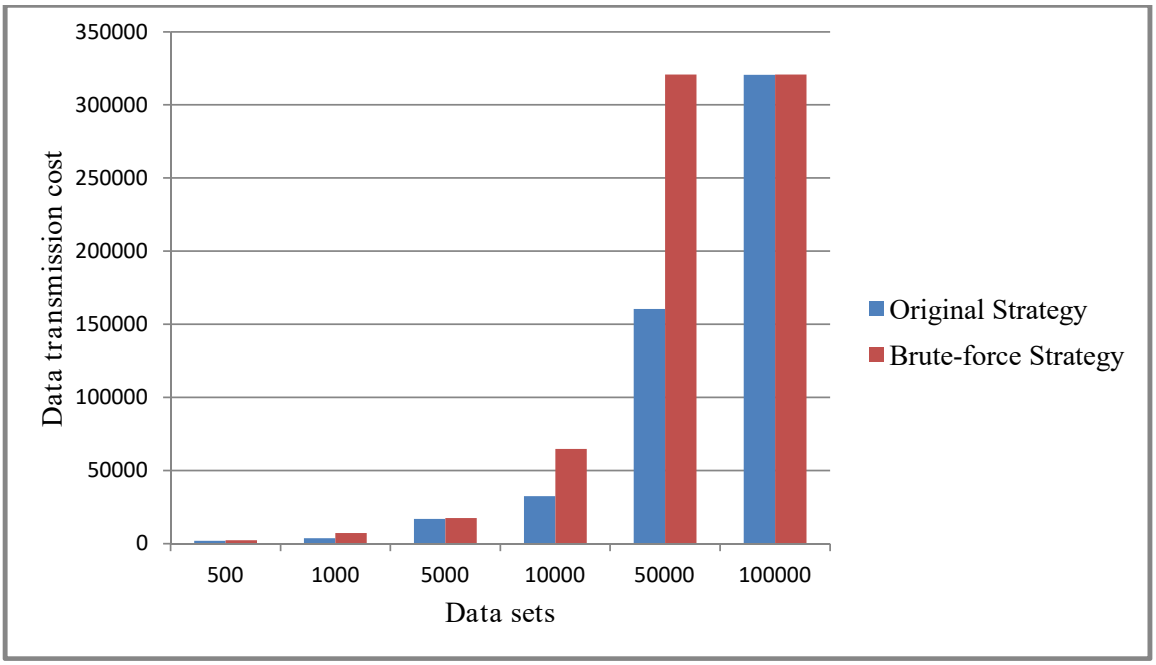


Figure 4.4: Average data transmission cost evaluation over static data

We also see for the 500, 5000 and 100000 point data sets that the improvement is not as significant. The data transmission cost is calculated depending on the query result: if the result file contains a large data set, that means the server has to send a large file to the client. Then, the transmission cost will be large. Similarly, if the result set contains a smaller file then the cost will be small. The query result can be small or large depending on

the identified superMBR and the number of residing points within that superMBR region for that query. The server with our strategy does not send the entire result file when the current results are the same as the previous results (which is shown in 4.4.2), but if those result files are smaller, it makes a small reduction to the average data transmission cost from the BF strategy (though BF strategy sends the entire result files for all 10 queries). As a consequence for the 500, 5000 and 100000 points data sets, the transmission cost is lower than the BF strategy but the reduction is not as significant.

4.4.2 Result-sending workload results

Table 4.1 shows the result sending workload of our original strategy and the BF strategy. We can see that when the server uses the BF method, it sends the entire result file to the client for all ten queries. That is, the server sends the full result every time, though in some cases different queries can have the same result.

From the table we can see that while using the original strategy, the server sends the entire file as the result for only six queries, for the 500, 1000, 5000, 50000, and 100000 points data sets. In case of the 10000 points data set, it sends the entire result file for seven queries. For the rest of the queries, the server sends only a message indicating that the result is same as the previous result, which reduces the server workload. It not only reduces the sending workload, but also reduces the data transmission cost and result sending time of our strategy.

Table 4.1: Result sending workload over static data

Data sets	Result sending workload (i.e., the number of times entire result set has to be sent to client)	
	Original strategy	Brute-force strategy
500	6	10
1000	6	10
5000	6	10
10000	7	10
50000	6	10
100000	6	10

Data sets	Response time (ms)				
	Original Strategy (OS)	Improved Strategy (IS)	Brute-force Strategy (BF)	BF/OS (%)	BF/IS (%)
500	291300	258900	294700	1.153716	12.14795
1000	1139000	1102800	1167300	2.424398	5.525572
5000	5402900	5277500	5396900	-0.11117	2.212381
10000	19693000	17140400	19950000	1.288221	14.08321
50000	70260500	66483400	76978700	8.727349	13.63403
100000	73084600	71379800	80552100	9.270398	11.38679

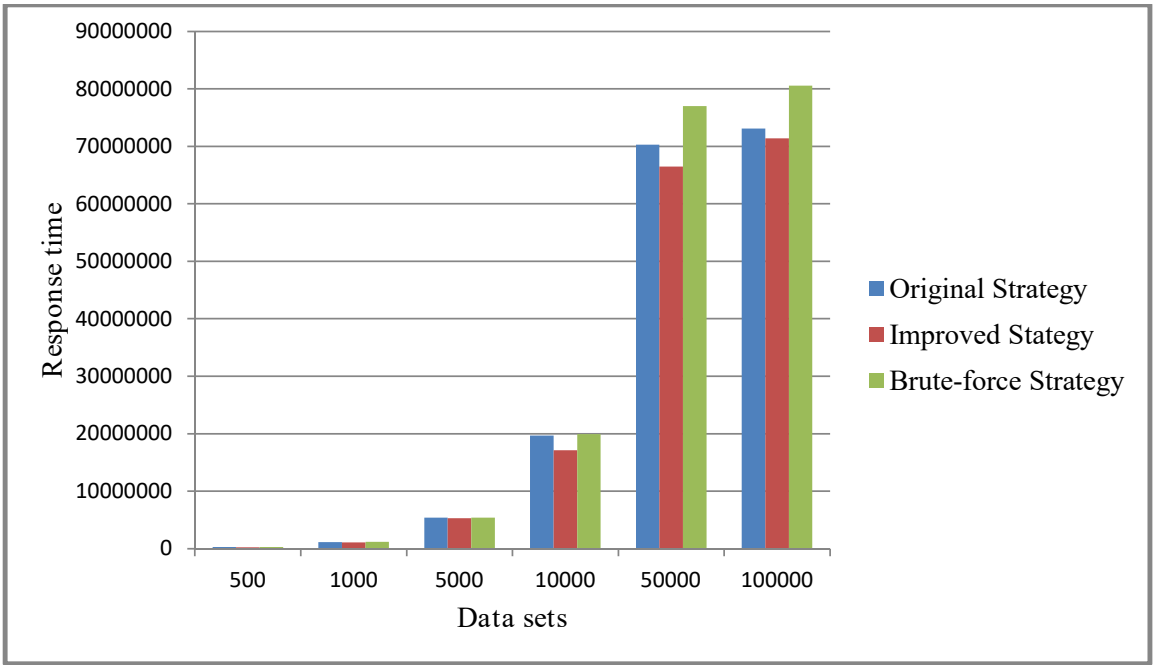


Figure 4.5: Average response time evaluation over static data

4.4.3 Response time results

Figure 4.5 shows the average query response time results of our original strategy, improved strategy and the BF strategy, for the six static data sets. In our original and improved strategy, the server does not send the same result file again to the client. Thus these strategies can reduce the query response time by lowering the sending time. Figure 4.5 also shows the percentage (%) of improvements of the average query response time in our strategies over the BF strategy.

We can see that for the 500, 1000, 10000, 50000 and 100000 points data sets, the query response time is lower in the original strategy than the BF strategy. In our original strategy, for each query the server compares the generated result with the previous result before sending an outcome to the client. The query result may contain a large set of points and in those cases the comparison time adds additional overhead. On the other hand, in the BF strategy, the server generates the result and then sends the result to the client. In this way, sometimes the BF strategy shows lower response times (in our case, for the 5000 points data set) because it sends the result while our original strategy must compare the result first.

Our improved strategy significantly reduces the file comparison overhead, which can reduce the response time. From Figure 4.5, we can see that the query response time is lower than the other two strategies. This is because our improved strategy monitors the changed points in the data set and only compares the changed point(s) to see whether this affects the result. So, in this strategy the server has to compare only the changed points which is generally much less than the entire result file.

4.5 Scenario 2: Moving client issues query on moving data.

In this experiment, the client issues ten region queries on moving data sets (discussed in Section 4.2). Here, we also compare our original strategy and the BF strategy with respect to the data transmission cost and result-sending workload. We compare the query response time of our improved strategy with both the original and BF strategies.

4.5.1 Data transmission cost results

Figure 4.6 shows the result of the data transmission cost evaluation. From the figure we can see that the data transmission cost of our original strategy is lower than the BF strategy, although not as low as for the static data set. The figure also shows the percentage (%) of the average data transmission cost improvement of our original strategy over the BF strategy.

For the moving data sets, the results of two queries may be different even though the

Data sets	Data transmission cost		
	Original Strategy (OS)	Brute-force Strategy (BF)	BF/OS (%)
500	2274.3	2304	1.289062
1000	7093.5	7318.4	3.073076
5000	16975.6	17436.8	2.644981
10000	64478.2	64619.9	0.219282
50000	320435.7	320787.2	0.109574
100000	320582.9	320822.4	0.074652

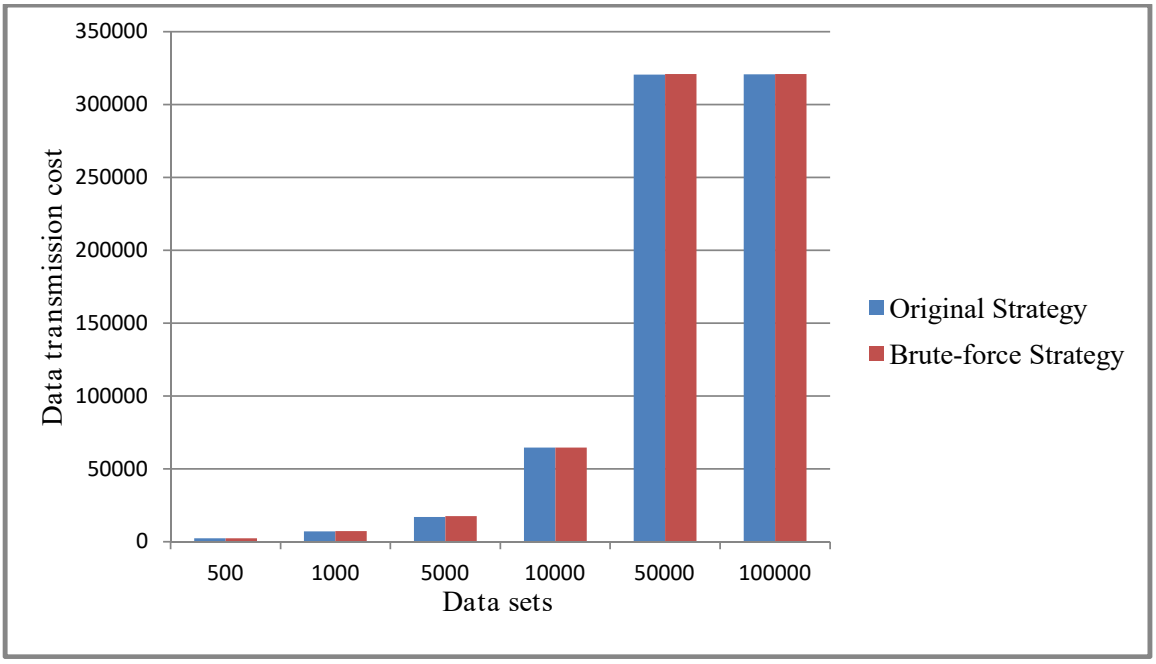


Figure 4.6: Average data transmission cost evaluation over moving data

superMBRs are the same. This is because, within the same superMBR region, some points may change their location, some may move out of that region, or some new points may appear in the region. In those situations, the point set within the superMBR changes and the server using our strategy needs to send the entire result file to the client even though the superMBRs are the same. So, for some individual queries, the data transmission cost of our strategy is the same as the BF strategy. Nonetheless, our original strategy has a lower average data transmission cost in comparison with the BF strategy because there still exist some queries that have the same superMBRs and no moving points within that region.

Therefore, if the point(s) are moving outside of the superMBR region, they do not affect the region.

4.5.2 Result-sending workload results

Table 4.2 shows the result-sending workload of the original and BF strategies for the six moving data sets. From the table we see that using our strategy, the server sends the entire result file nine times for the 500 and 1000 points data sets, seven times for the 50000 and 100000 points data sets, six times for the 5000 points data set, and eight times for the 10000 points data set, among all ten region queries. Although the result-sending workloads are higher here, there are still some savings that are achieved. This reduces the response time for our strategies.

Table 4.2: Result sending workload over moving data

Data sets	Result sending workload (i.e., the number of times entire result set has to be sent to client)	
	Original strategy	Brute-force strategy
500	9	10
1000	9	10
5000	6	10
10000	8	10
50000	7	10
100000	7	10

4.5.3 Response time results

Figure 4.7 shows the average query response time of our original strategy, improved strategy and the BF strategy over the six moving data sets. From the figure, we see that the query response times of our original strategy are sometimes about the same or even higher than the BF strategy. This is because of the same comparison overhead as mentioned in Section 4.4.3. Our improved strategy reduces the comparison overhead by monitoring the changed points only, and thus produces the lowest average query response times for all six

data sets among the three strategies. The percentage (%) of the average query response time improvements in our strategies over the BF strategies are also shown in Figure 4.7.

Data sets	Response time (ms)				
	Original Strategy (OS)	Improved Strategy (IS)	Brute-force Strategy (BF)	BF/OS (%)	BF/IS (%)
500	626700	535100	566900	-10.54859764	5.60945493
1000	1795100	1743700	1946200	7.763847498	10.40489154
5000	4952200	4771900	4810900	-2.93708038	0.810659128
10000	19566100	19159800	19167200	-2.081159481	0.038607621
50000	50834800	49923800	50467300	-0.728194296	1.076934966
100000	83817000	73528800	103953200	19.37044747	29.2674011

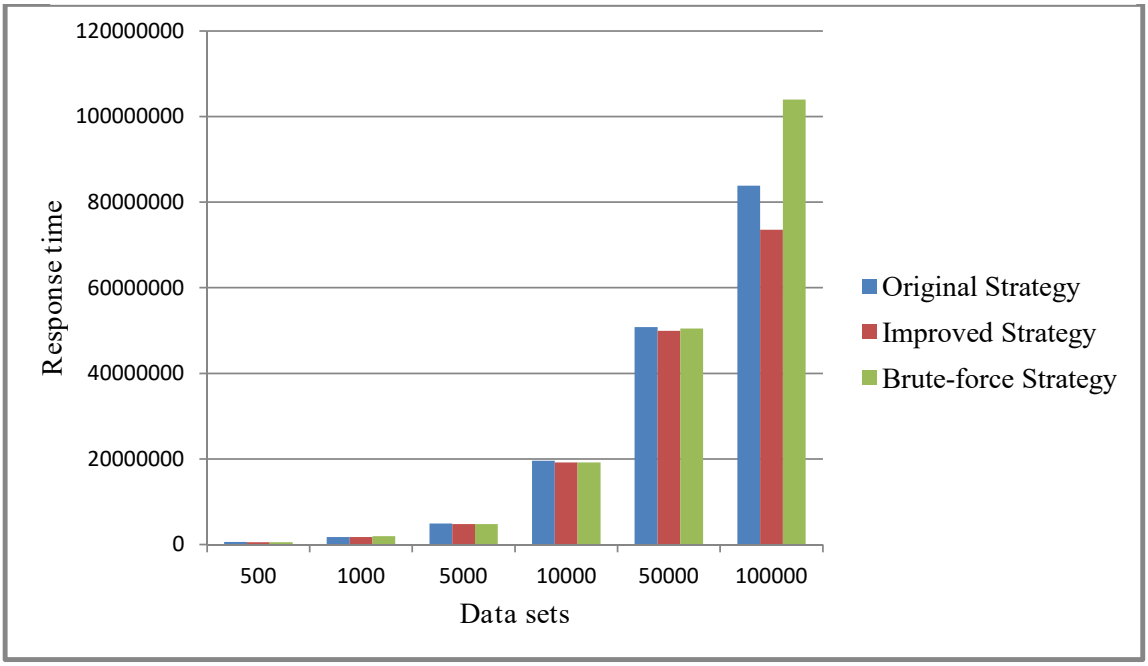


Figure 4.7: Average response time evaluation over moving data

4.6 Scenario 3: Moving client issues query on moving data, where the number of moving points on the data set varies

In the third experiment, we investigate the performance of our strategies by varying the number of moving points. Here each data set contains 1000 points, but a different number

of moving points (described in Section 4.2). As before, we compare the data transmission cost and the result sending workload of our original strategy with the BF strategy only. Then we evaluate the query response time performance of our improved strategy and compare it with the original and BF strategies.

Moving points	Data transmission cost		
	Original Strategy (OS)	Brute-force Strategy (BF)	BF/OS (%)
1	7093.5	7318.4	3.073076
2	7002.1	7318.4	4.321983
3	7008.5	7328	4.359989
4	7008.5	7328	4.359989
5	7008.5	7328	4.359989

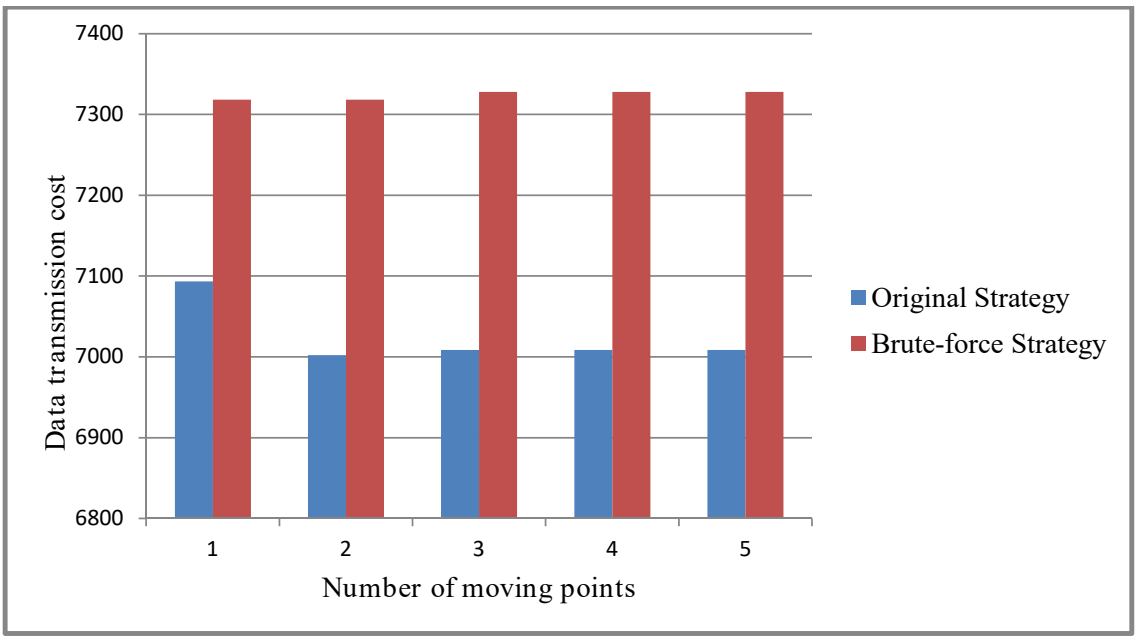


Figure 4.8: Average data transmission cost evaluation over varying moving points

4.6.1 Data transmission cost results

Figure 4.8 shows the average data transmission cost of the original and BF strategies for the five data sets with five different (i.e., 1, 2, 3, 4, 5) moving points. From the figure, we see that the data transmission cost is almost the same for different numbers of moving points

and still lower than the BF strategy. If any number of points move, it does not significantly affect the data transmission cost. This is because if the residing points inside the superMBR region only internally change their location, then the data transmission cost will remain the same. Again, if any one point within the superMBR region moves, it affects the result, and the server with our strategies sends the entire new result to the client. That means that once one point causes change in the result, the server must send the entire result, no matter if other points change or not.

4.6.2 Result sending workload results

The result sending workload of the original and BF strategy for the five 1000 points data sets with different numbers of moving points is shown in Table 4.3. From the table, we see that using our strategy the server sends the entire result file nine times when only one points moves, and seven times when 2, 3, 4 or 5 points move in the data set. If there is a change in the new result compared with the previous result, the server in our strategy must send the entire result to the client. That means that once one point causes a change in the result, the server has to send the entire result, it does not matter if other points change or not. This is why the result sending workload does not significantly increase with the increasing number of moving points.

Table 4.3: Result sending workload when 1 to 5 points moves in the data sets

Number of moving points	Result sending workload (i.e., the number of times entire result set has to be sent to client)	
	Original strategy	Brute-force strategy
1	9	10
2	7	10
3	7	10
4	7	10
5	7	10

Moving points	Response time (ms)				
	Original Strategy (OS)	Improved Strategy (IS)	Brute-force Strategy (BF)	BF/OS (%)	BF/IS (%)
1	1677900	1631500	1711800	1.980371539	4.690968571
2	1855100	1488100	1609300	-15.27372149	7.531224756
3	1793600	1443500	1673000	-7.208607292	13.71787209
4	1730200	1616200	1658100	-4.348350522	2.526988722
5	1899600	1652100	1784300	-6.461917839	7.409067982

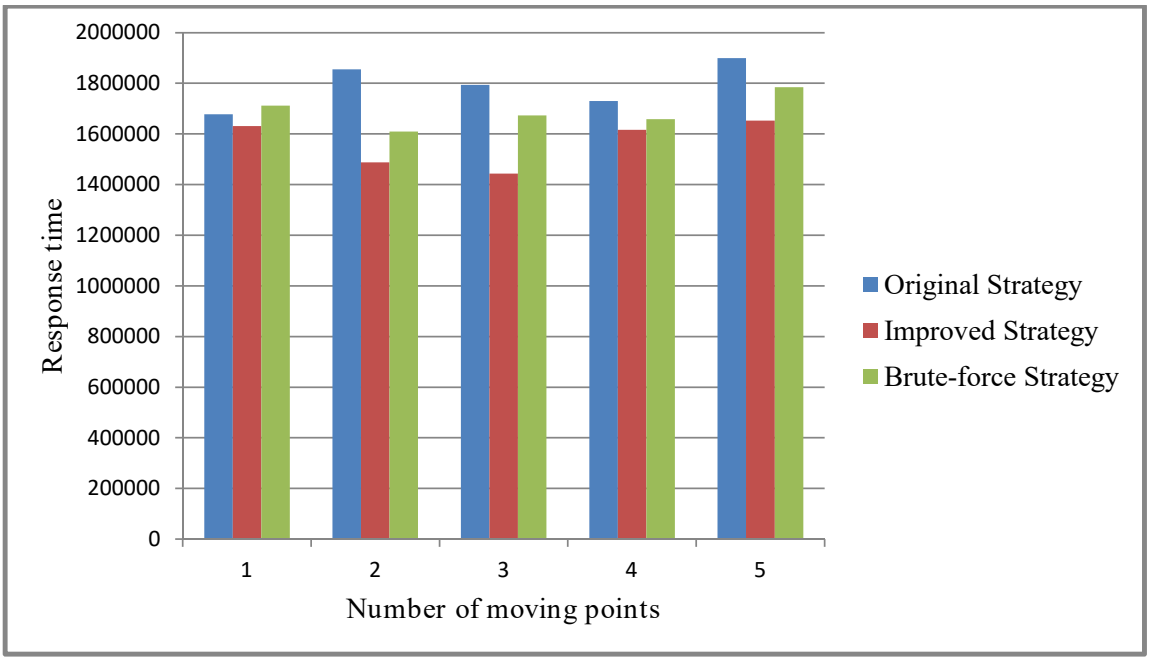


Figure 4.9: Average response time evaluation over different moving points

4.6.3 Response time results

Figure 4.9 shows the average query response time of our original, improved and BF strategies. From the figure we see that our improved strategy has the lowest response time among the three. The BF strategy sends the result file every time, so its response time is higher than the improved strategy. Our original strategy still has occasional higher response times because every time it does not send the entire result file, it must compare the new result with the previous result which increases the response time. The improved strategy reduces this comparison overhead by monitoring and comparing only the changed points

within the superMBR region. Figure 4.9 also shows the percentage (%) of average query response time improvement of our strategies over the BF strategy.

4.7 Conclusion

Chapter 3 proposed two continuous region query processing strategies for location-based services. For the performance evaluation we compare our strategies with a Brute-force strategy. This chapter presents the experimental evaluation and comparison of these strategies.

Chapter 5

Conclusion

5.1 Conclusion

We propose two query processing strategies for location-based services. Both strategies are for continuous region query processing over moving point sets, where the query issuer object or client is also moving. In our first strategy, the server compares the current result set with the previous result in order to identify any change in the new result. If the current result is different from the previous query result, the server sends the entire result file. Otherwise it sends a message indicating that the current result is the same as the previous result. In some cases for the first strategy, when the result contains a very large data set, then comparing all of the data between the large result sets causes significant overhead for the server, and increases the query response time. In order to avoid this problem, our improved strategy uses monitoring to track any points that have moved. Instead of comparing all points in the result set, our improved strategy tests only the moved point(s) to make a decision as to whether the current result is different from the previous query result or not.

We evaluate our strategies in three different scenarios. In the first scenario, the moving client issues queries on static data. In the second scenario, the moving client issues queries on moving data. And in the third scenario, the moving client issues queries on moving data, but in this case the number of moving points on the data set varies.

The experimental results show that our strategies achieve significant improvements over the brute-force strategy. We compare our first strategy with a brute-force strategy and found that our strategy can significantly reduce the server workload and data transmission cost

over the brute-force method. Our original strategy achieves up to 49.98% of data transmission cost performance improvement for static data and up to 3.07% improvements for moving data, over the brute-force strategy. The data transmission cost and server workload of our improved strategy are the same as the original strategy. We compare our improved strategy with the original strategy and brute-force strategy. The experimental results show that the improved strategy achieves lower query response time than the original and brute-force strategies. Our improved strategy achieves up to 14.08% of response time performance improvement for static data and up to 29.26% improvement for moving data, as compared to the brute-force strategy.

5.2 Future work

In our proposed strategies, we use a spatial access method which is mqr-tree. In the future, we will compare our strategies with other query processing strategies which use other spatial index method (i.e., R-tree) instead of the mqr-tree.

We are also planning to propose a client side processing strategy so that the client can locally process a query while moving within the validity region (i.e., superMBR region). It should be noted that the query issuer object and points of the data set are moving. Now, when a client needs to process any region query that resides in the previous superMBR region, the client request for only the current record of changed/moved point(s) within that superMBR region from the server, instead asking for the entire query result. After receiving the record of moved point(s), the client will process the query from the previous result set and modify the query answer according to the moved point(s) record. In this way, when the superMBRs of consecutive queries are the same, after processing and sending the result of the first query to the client, the server only sends the moved point(s) record instead of processing the next query. This will further reduce the server side query processing workload and also further reduce the query response time.

Bibliography

[1] Gloria Bordogna, Marco Pagani, Gabriella Pasi, and Giuseppe Psaila. Managing uncertainty in location-based queries. *Fuzzy sets and systems*, 160(15):2241–2252, 2009.

[2] Huiping Cao, Shan Wang, and Lingwei Li. Location dependent query in a mobile environment. *Information Sciences*, 154(1):71–83, 2003.

[3] Reynold Cheng, Kam-Yiu Lam, Sunil Prabhakar, and Biyu Liang. An efficient location update mechanism for continuous queries over moving objects. *Information Systems*, 32(4):593–620, 2007.

[4] Ping Fan, Guohui Li, and Ling Yuan. Continuous k-nearest neighbor processing based on speed and direction of moving objects in a road network. *Telecommunication systems*, 55(3):403–419, 2014.

[5] Yunjun Gao and Baihua Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 577–590. ACM, 2009.

[6] Yunjun Gao, Baihua Zheng, Gang Chen, Chun Chen, and Qing Li. Continuous nearest-neighbor search in the presence of obstacles. *ACM Transactions on Database Systems (TODS)*, 36(2):9, 2011.

[7] Yunjun Gao, Baihua Zheng, Gencai Chen, Qing Li, and Xiaofa Guo. Continuous visible nearest neighbor query processing in spatial databases. *The VLDB Journal*, 20(3):371–396, 2011.

[8] Manish Gupta, Manghui Tu, Latifur Khan, Farokh Bastani, and I-Ling Yen. A study of the model and algorithms for handling location-dependent continuous queries. *Knowledge and information systems*, 8(4):414–437, 2005.

[9] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[10] Liang Hong, Gang Zhou, Bo Liu, and Sang Son. Continuous location dependent queries in mobile wireless sensor networks. *Wireless personal communications*, 68(1):153–173, 2013.

[11] Yu-Ling Hsueh, Roger Zimmermann, and Wei-Shinn Ku. Efficient location updates for continuous queries over moving objects. *Journal of Computer Science and Technology*, 25(3):415–430, 2010.

[12] Jiun-Long Huang and Chen-Che Huang. A proxy-based approach to continuous location-based spatial queries in mobile environments. *Knowledge and Data Engineering, IEEE Transactions on*, 25(2):260–273, 2013.

[13] Yuan-Ko Huang, Chao-Chun Chen, and Chiang Lee. Continuous k-nearest neighbor query for moving objects with uncertain velocity. *GeoInformatica*, 13(1):1–25, 2009.

[14] Yuan-Ko Huang, Zhi-Wei Chen, and Chiang Lee. Continuous k-nearest neighbor query over moving objects in road networks. In *Advances in Data and Web Management*, pages 27–38. Springer, 2009.

[15] Yuan-Ko Huang and Chiang Lee. Efficient evaluation of continuous spatio-temporal queries on moving objects with uncertain velocity. *Geoinformatica*, 14(2):163–200, 2010.

[16] Dick Hung, Kam-Yiu Lam, Edward Chan, and Krithi Ramamritham. Processing of location-dependent continuous queries on real-time spatial data: The view from retina. In *DEXA Workshops*, pages 961–965. Citeseer, 2003.

[17] Sergio Ilarri, Carlos Bobed, and Eduardo Mena. An approach to process continuous location-dependent queries on moving objects with support for location granules. *Journal of Systems and Software*, 84(8):1327–1350, 2011.

[18] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-dependent queries in mobile contexts: Distributed processing using mobile agents. *Mobile Computing, IEEE Transactions on*, 5(8):1029–1043, 2006.

[19] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys (CSUR)*, 42(3):12, 2010.

[20] Glenn S Iwerks, Hanan Samet, and Kenneth P Smith. Maintenance of k-nn and spatial join queries on continuously moving points. *ACM Transactions on Database Systems (TODS)*, 31(2):485–536, 2006.

[21] Mohammad R Kolahdouzan and Cyrus Shahabi. Alternative solutions for continuous k nearest neighbor queries in spatial network databases. *GeoInformatica*, 9(4):321–341, 2005.

[22] Wei-Shinn Ku, Roger Zimmermann, and Haixun Wang. Location-based spatial query processing with data sharing in wireless broadcast environments. *Mobile Computing, IEEE Transactions on*, 7(6):778–791, 2008.

[23] Young-Mo Kwon, Harim Jung, and Yon Dohn Chung. Monitoring continuous k-nearest neighbor queries in the hybrid wireless network. *Journal of Zhejiang University SCIENCE C*, 12(3):213–220, 2011.

[24] Kam-Yiu Lam and Özgür Ulusoy. Adaptive schemes for location update generation in execution location-dependent continuous queries. *Journal of Systems and Software*, 79(4):441–453, 2006.

[25] Ken CK Lee, Brandon Unger, Baihua Zheng, and Wang-Chien Lee. Location-dependent spatial query containment. *Data & Knowledge Engineering*, 70(10):842–865, 2011.

[26] Chow-Sing Lin and Shiou-Yun Wu. Processing directional continuous range queries for mobile objects on road networks. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, pages 330–335. IEEE, 2014.

[27] Fuyu Liu. Query processing in location-based services. 2010.

[28] Fuyu Liu and Kien A Hua. Moving query monitoring in spatial network environments. *Mobile Networks and Applications*, 17(2):234–254, 2012.

[29] Marc Moreau and Wendy Osborn. mqr-tree: A 2-dimensional spatial access method. *arXiv preprint arXiv:1212.1469*, 2012.

[30] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of spatial queries in wireless broadcast environments. *Mobile Computing, IEEE Transactions on*, 8(10):1297–1311, 2009.

[31] Kyriakos Mouratidis and Dimitris Papadias. Continuous nearest neighbor queries over sliding windows. *Knowledge and Data Engineering, IEEE Transactions on*, 19(6):789–803, 2007.

[32] Kyriakos Mouratidis, Dimitris Papadias, and Marios Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 634–645. ACM, 2005.

[33] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *Proceedings of the 32nd international conference on Very large data bases*, pages 43–54. VLDB Endowment, 2006.

[34] Wendy Osborn and Annika Hinze. Tip-tree: A spatial index for traversing locations in context-aware mobile access to digital libraries. *Pervasive and Mobile Computing*, 15:26–47, 2014.

[35] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005.

[36] Kwangjin Park. Efficient data access for location-dependent spatial queries. *Journal of Computer Science and Technology*, 29(3):449–469, 2014.

[37] KwangJin Park, MoonBae Song, and Chong-Sun Hwang. Adaptive data dissemination schemes for location-aware mobile services. *Journal of Systems and Software*, 79(5):674–688, 2006.

[38] Shashi Shekhar and Sanjay Chawla. *Spatial databases: a tour*, volume 2003. prentice hall Upper Saddle River, NJ, 2003.

[39] Haojun Wang and Roger Zimmermann. Processing of continuous location-based range queries on moving objects in road networks. *Knowledge and Data Engineering, IEEE Transactions on*, 23(7):1065–1078, 2011.

[40] Baihua Zheng, Wang-Chien Lee, Ken C Lee, Dik Lun Lee, and Min Shao. A distributed spatial index for error-prone wireless data broadcast. *The VLDB JournalThe International Journal on Very Large Data Bases*, 18(4):959–986, 2009.