# Moving Range Query Processing in Spatial Databases

by

**Haidar AL-Khalidi, MCompSc**


**Thesis**

Submitted by Haidar AL-Khalidi

for fulfilment of the requirements for the degree of

**Doctor of Philosophy (0190)**

Supervisor: A/Professor David Taniar

Associate Supervisor: Dr. John Betts

**Clayton School of Information Technology**
**Monash University**

December, 2014

To All People I Love

# Contents

# List of Tables

# List of Figures

# Moving Range Query Processing in Spatial Databases

Haidar AL-Khalidi, MCompSc
[redacted]
Monash University, 2014

Supervisor: A/Professor David Taniar
[redacted]
Associate Supervisor: Dr. John Betts
j[redacted]

**Abstract**

Recent developments in mobile communications have brought dramatic and fundamental changes to the modern world. These developments have resulted in a great demand for applications that integrate geographic locations and services to fulfill the user's needs. Different types of spatial queries are used in such applications, however, most studies consider the moving range query to be the most common, and frequently used. The moving range query is used to find all objects of interest within a given radius while the user who invokes the query is moving. During the last decade, some studies in moving range queries considered Euclidean geometry, where the distance between two objects is determined by their relative position in space. Some other studies considered network distance, wherein the trajectory between two objects is specified by the underlying network.

The main objective of all the previous studies is to process moving range queries efficiently. However, most of these studies focus on index efficiency and less effort has been made to address the issues of moving query updating and optimisation, which are crucial factors affecting system performance. In

this thesis, we attempt to investigate this possibility by proposing four new processing techniques.

First, we introduce a new "*approximate moving range query*" to optimise the query process in two ways: *i*) we reduce the amount of time needed to obtain the results; and *ii*) we give the users alternative options in order to avoid another search(es) when searching an empty result or a massive number of objects in the result. The result of our experiments show that our techniques are successful in terms of reducing the number of split points and false hits. Also, the approximate results are of a high quality compared to the exact results. Furthermore, our algorithms reduce the number of communications between the mobile device and the databases server, indicating their performance is better in terms of lower search time and improved search accuracy.

Second, we present a technique to deal with the moving range query called "*safe region*". The high computation and communication costs of monitoring and updating the location of the moving range query needs to be considered, as the calculation of the range query needs to be re-evaluated whenever the query moves. Our aim is to avoid any communication between the query and the server while the query moves within the specified safe region. We have extended the size of the safe region to reduce the amount of supplementary communication. Our main objective is to reduce the need for continuous monitoring of the query, and eliminate the need for the user to follow a defined path.

Third, we propose a linear model called "*monitoring moving range query*" to monitor moving queries inside a safe region. Our technique gives the users the ability to monitor themselves and inform the server when leaving the safe region. This will give the user more privacy and reduce the load on the server. Also, our technique will eliminate the need for the user to follow a defined path. We use the time and the concept of the safe region together, hence, if the

query makes a sudden turn, the result will not be affected because the query will still be located inside the safe region.

Finally, we introduce a novel query processing technique for the moving range query in spatial network databases, called "*lookforward moving range query*". Our technique is related to Euclidean and network distance to retrieve related objects and at the same time to exclude unrelated ones. Our technique can distinguish between the significant important objects and the minor important objects inside the range query. Our technique improves the selectivity of the filter step to reduce the number of candidate objects, and consequently, minimise the number of communications between the mobile device and the database server. The lookforward moving range query achieves a better running time and delivers a better performance having fewer split points than the original moving range query as shown on our experiments.

# Moving Range Query Processing in Spatial Databases

**Declaration**

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Haidar AL-Khalidi
December 14, 2014

# Acknowledgments

I would like to thank everyone who helped to make this possible. It has been an incredible journey of self-discovery, and I love every last one of you...

First of all, I would like to express my special appreciation and thanks to my supervisor, Associate Professor Dr. David Taniar. You have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Many thanks for your patience and valuable advice during my candidature. Your advice on both research as well as on my career have been priceless.

I would also like to thank my co-supervisor, Dr. John Betts, for helpful advice and insight. His valuable editorial and technical advice have been essential to the completion of my Ph.D. thesis.

In addition, a thank you goes to Professor Maytham Safar (Kuwait University). I have been extremely lucky to have a person like you who cared so much about my work, and who responded to my questions and queries so promptly.

I would especially like to thank all the staff of Clayton School of Information Technology at Monash University. All of you have been there to support me whenever I needed any help during my Ph.D. program.

I thank all my wonderful friends in our spatial database reading group Dr. Kefeng Xuan, Dr. Geng Zhao, Dr Thao P. Nghiem, Dr. Sultan Alamri, Kiki Maulana, Dr. Kinh Nguyen and Dr. Muhammad A. Cheema.

A special thanks to my family, including my in-laws. Words cannot express how grateful I am to my mother, and father for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far. Many thanks for your support, encouragement and patience during three and a half years of my Doctorate in Information Technology.

At the end, special thanks goes out to my lovely wife Zainab, who spent sleepless nights with me and was always my support in the moments when there was no one to answer my queries. She inspired me and provided constant encouragement during the entire process, as well as continuously proofing my document.

Haidar AL-Khalidi

*Monash University*

*December 2014*

# Chapter 1

## Introduction

# 1

# Introduction

Over the last two decades, spatial databases have received increasing interest due to their important role in many modern applications, such as Geographic Information Systems (GIS), multimedia databases, urban planning, and traveller information systems (Xuan, Zhao, Taniar, Safar, & Srinivasan, 2011; Bustos & Navarro, 2009; Safar, 2008). The key characteristic that makes this type of database a powerful tool is its ability to manipulate the data, instead of only storing and representing the data. The most basic form of such manipulation is answering queries related to spatial properties of data (Corral et al., 2000). The response to a query returns all objects of interest that satisfy the selection conditions, and are close to the given query. There are many types of queries in spatial and mobile databases proposed and studied in the last decade. One of the most important and frequently used ones among these queries is the moving range query.

Moving range query (Continuous range query) is defined as the retrieval of information of objects of interest while the query moves, and continuous monitoring of the change of the query results over a certain time period. For

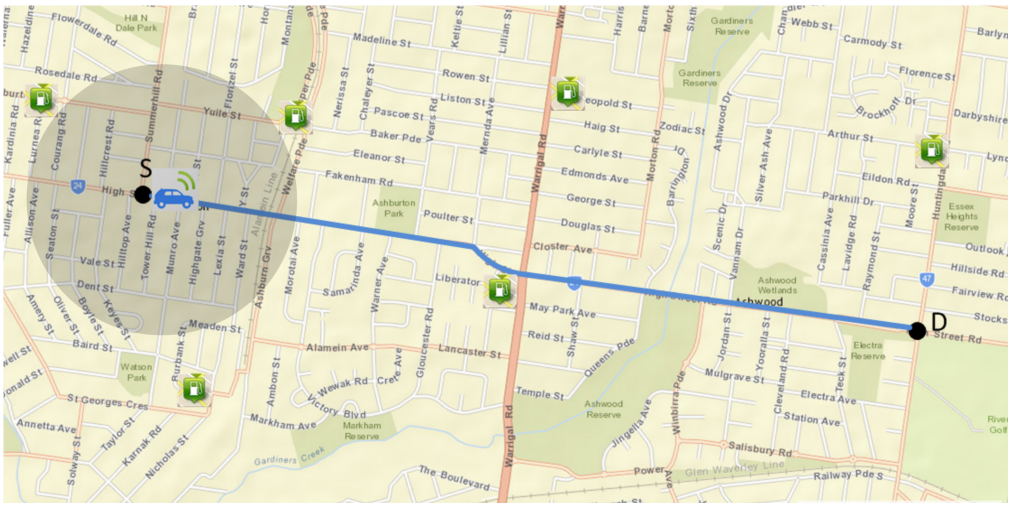



Figure 1.1: Example of moving range query

the rest of the thesis term of the moving range query will be used instead of the term of continuous range query.

**Definition:** Given a set of spatial objects $P = [p_1, p_2, ..., p_n]$ and a query path $q = [S, D]$, where $S$ and $D$ are start and distention point of the path ($q$) respectively. All objects within a radius $e$ from the path (i.e. $D_E(q, p_i) \leq e$) will be in the result list of the moving range query.

In Figure 1.1 the user invokes a range query at location $S$ looking for all objects of interest within a specific distance while s/he moves to destination $D$. For example, a driver of a moving car posted the following query: "Give me the locations and names of the petrol stations offering unleaded fuel for less than \$1.50 per litre within $2km$". In this example, the central location of the query is the car on the move and the target objects are buildings within $2km$ with respect to the location of the car on the move.

The moving range query raises challenges to researchers because the movement of the central location consequently changes the query result. Several studies have been conducted to address these challenges. However, many limitations still need to be addressed with a demand for further improvements to be

made. In Section 1.1, we clarify the objectives of this thesis. In Section 1.2 the main problems of the existing approaches are explained. Section 1.3 presents our contributions. Finally, Section 1.4 presents the outline of the thesis.

## 1.1 Research Objectives and Motivation

With the growing popularity of mobile navigation and Global Position System (GPS) in mobile information systems, the efficient processing of a moving range query has been of increasing interest. The traditional range query is meant to be static, where the query is based on the current users' location (Bustos & Navarro, 2009; Xuan, Zhao, Taniar, Rahayu, et al., 2011) and the results are based on a static range query. Once the results are assembled by the server, then they will be sent to the query user. But, when users are moving, the static range query will not be adequate, therefore, the moving range query has been developed. Over the years, researchers have used two types of distance measurement to locate objects of interest , namely *Euclidean distance* measurement and *road network distance* measurement. Measuring using Euclidean spaces means that the distance between two objects is determined by their relative position in space (i.e., a straight and direct distance between two spatial objects). On the other hand, measuring using road network distance means considering the actual network distance between two objects (the length of the shortest road connection between them on the map) (Zhao et al., 2014). This type of measurement has been used because in real life there should be an underlying road network where spatial objects are located. In this thesis, we cover both of these types of measurements in processing moving and static range query processing.

Our motivation in this study is based on the reality that: most existing works mainly focus on index efficiency, while less effort is made to address the

limitations of query processing and updating. However, these are crucial factors affecting system performance. As well, existing works in processing moving range queries require a vast amount of network distance computation along with continuous online communications between the user and the database server to update the query result. In addition to a careful consideration of the query path direction, in some cases, ignoring the direction from the query point to the objects of interest may result in having the passed-by objects of interests in the result list.

Thus, the main objective of our research is to:

- investigate new ways to process the moving range query more efficiently and accurately.
- present new range queries that minimise the communication cost of tracking the moving query.
- develop a new model to obtain the result in a short time with high quality guaranteed answers.
- construct a new technique to reduce the need for continuous monitoring of the query, and eliminate the need for the user to follow a defined path.

## 1.2 Major Problems

In this thesis, we concentrate on finding adequate solutions to the existing limitations of processing and updating moving range queries to satisfy two criteria: the computational *time* and the *cost* of the query search. Several studies have been conducted during the last decade to reduce the computational time and the cost of the search, however, there is a demand for further improvement which we have framed as several problems:

- **Problem 1**: The excessive amount of time needed to obtain the results. Many false hits are retrieved when the query radius *e* is expanded, which makes processing moving range query time consuming. Also, the density of the objects and the performance of the range query technique are inversely proportional, while the set size of the answer is directly proportional to the size of the spatial region.

- **Problem 2**: Two issues arise when *critical objects* (i.e., objects with minimum distance to the query path equal or almost equal to *e*) enter a query boundary for only short time: *i*) relevant objects, name *false misses*, may not be included in the results, and *ii*) the user does not have time to react when critical objects are present for only a few seconds.

- **Problem 3**: Split points (split nodes) are the locations where query results are updated. The number of split points is twice the number of objects of interest (Xuan, Zhao, Taniar, Safar, & Srinivasan, 2011), the effect of which is a substantial increase in the communication traffic between the mobile user and the server.

  When the range query results change, the database server informs the mobile user, so a need for continuous online communication is created in order to update the user with the changed results. Also, it is important to note that in wireless environments, mobile devices experience limited bandwidth and low-quality communication (Chow et al., 2009).

- **Problem 4**: Communication between a user and a random moving query is significant and expensive with real-time updating of query answers. In some applications delays result in outdated answers and in situations with a high density (500 objects) of objects and continuous updating the

editing is so fast (less than five seconds) that it reduces the user's ability to make decisions. Also, with inaccurate prediction of the location of a random moving query frequent updates of its location are required, so it becomes important to reduce the number of location updates. When a user, however, is following a predicted path the client (user) should not need to inform the server of its location.

## 1.3 Thesis Contributions

This thesis specifically aims to improve the processing of the moving range query, develop a new model to obtain the results in a short time and with guaranteed high quality. In this thesis, we also introduce a new query algorithm to reduce the location updates and the communication between the query and the server. Furthermore, we construct a new technique to reduce the need for continuous monitoring of the query, and eliminate the need for the user to follow a defined path.

### 1.3.1 Approximate Range Query

Mobile communication technologies have brought significant change to everyday life. One of the growing applications is mobile navigation, which helps users navigate crowded roads using the best route and gives answers to location queries (Zhao et al., 2014). The execution of users' queries, however, comes at a high cost.

A quest to improve the efficiency of query search techniques has led to investigation of the concept of approximate (Arya et al., 2009). In real life exactness in location information of an object is not required for all applications

(Petkova et al., 2009). For example, a traveller wants to know the arrival time of the next train. The exact location is not important.

Also, in some applications such as GIS and cellular networks, the comprehensive data and the number of objects of interest in addition to the costly I/o operations and response time, all these factors contribution to the necessity of examining the approximate techniques in order to obtain an efficient results (Corral et al., 2002).

The approximation concept is used in this thesis to improve the processing and efficiency of range queries (i.e., static and moving). Novel techniques are introduced to demonstrate that the approximate results can be obtained much faster and at less cost than the exact results. To date no studies have been conducted using an approximate static range query relying on a distance based query and using a moving range query in the context of approximation.

Our research presented here was published in (Al-Khalidi et al., 2011; AL-Khalidi et al., 2013; AL-Khalidi, Taniar, & Safar, 2013).

In spatial databases and using the static range query, search time is important and is affected by the number of retrieval objects and the size of the range query, which retrieves all objects in a range from a particular query point. The result takes an excessive amount of time and can return no result or a huge number of objects, thus necessitating another search. Also, repeated false hits (i.e., irrelevant objects as candidates) are investigated and are a waste of time. These problems can be addressed using the concept of approximation which provides valuable results much faster. Three novel approximate methods are introduced, namely, *Lowerbound* approximate static range query (*Lowerbound* ASR) and *Upperbound* approximate static range query (*Upperbound* ASR) and approximate static range query (ASR) to enhance the performance of the range query by using different factors to bound the range search.

The moving range query has the problem of a large number of communications between the database server and the mobile device. In areas with a high density of objects and fast updating of the search result, users have some difficulty in making decisions. These problems are addressed once again using the approximation concept. Two approximate moving range query techniques are proposed: reduction of the range search to reduce the number of critical objects and minimise search time; and reduction of the number of split points to reduce the number of times that results are updated and the number of communications with the server.

Implementing range queries (static and moving) based on *spatial network databases*, namely Range Euclidean Restriction RER and Range Network Expansion RNE, requires lengthy calculation times to obtain the essential results. These queries need a massive number of calculations of the road work in all directions looking for objects of interest. The queries based on road network are similar to those queries based on Euclidean distance. They are also suffer from repeated false hits and a result list with a massive number of objects or with absolutely no objects. Therefore, we design two new methods, called *Approximate Range Euclidean Restriction (ARER)* and *Approximate Range Network Expansion (ARNE)* to eliminate the problems of the former methods (Range Euclidean Restriction RER and Range Network Expansion RNE). We also introduce another two methods for moving range query named *Range Search Minimisation* and *Split Points Minimisation*. In these techniques, we introduce *lowerbound* which minimises the actual range search to exclude the internal nodes that fall outside the *lowerbound*. We also improve the filter step to reduce the number of candidate objects and the number of communications with the server. The technique has better running time and performance, but with low false hits and reasonable false misses.

### 1.3.2 Safe Region in Moving Range Query

In the moving range query, the query is assumed to be constantly moving. Minimising the frequent updates of the query location and keeping low costs while monitoring the moving query are the two main challenges researchers have to face. We address these challenges and present new efficient methods using the safe region concept, called range safe region.

The safe region is an area where the set of objects of interest does not change as long as the query remains inside it. The aim of the safe region is to reduce the number of the query location updates by reducing the number of queries to the server. The main challenge is how to keep the query result up to date while the user is moving and how to reduce to the minimum the server's monitoring for the user. In response, we introduce three new types of safe regions called: Basic, Enhanced, and Extended. Monte-Carlo simulation is used to calculate the total area of the safe region.

This work was published in (Al-Khalidi et al., 2013b; AL-Khalidi, Taniar, Betts, & Alamri, 2013).

Our contributions can be summarised as follows: First, we propose a new technique to construct a safe region, termed the range safe region. The aim of this technique is to avoid any communication between query and server while the query moves within the specified safe region. Second, we extend the range safe region to reduce the amount of supplementary communication. Using these techniques reduces the need for continuous monitoring of the query, and eliminates the need for the user to follow a defined path. We use a discrete-event simulation method with a high degree of accuracy to calculate the area of the safe regions that have been formed by overlapping range objects. Finally, we compare the Basic, Enhanced and Extended safe regions when the query

is static and when it is moving. Our method reduces the amount of query monitoring by the server and prevents any communication costs while the query moves within its specified constructed safe region. The new method also allows a user to move within the safe region without revealing their location, thus preserving their privacy.

### 1.3.3 Monitoring Moving Range Query

The moving range query requests constant reporting of its results which extend from the registration of the query to its cancellation. This is called the effective period of the query. Over this time, the query results must be continuously updated even if the query conditions remain unaltered during the effective period (Shengsheng & Chen, 2011). To reduce the updating costs while the query moves continuously, the safe region concept has been proposed (AL-Khalidi, Taniar, Betts, & Alamri, 2013; Cheema et al., 2011; Cho et al., 2013; Mokbel et al., 2004), which allows the query to report its current location and to request a new result only when it exits its current safe region. These two factors have the potential to significantly reduce communication and computational overheads. Because the query does not communicate with the server once it enters its safe region, neither the query nor the server will be aware of when and from which direction the query will leave its assigned safe region. Consequently, the server would not be able to calculate a new safe region for the query to roam in or update the result list without delay.

In this thesis, we present our technique for continuously monitoring a range query inside the Extended Safe Region. A linear function is proposed to monitor the moving query within its safe region. Since the Extended Safe Region has an irregular shape, it is necessary to find a method to monitor the query inside it.

Our method reduces the costs associated with communications in client-server architectures because an update of the location will be reported only when the query leaves its assigned safe region or upon the server's request. Computational results show that our method is successful in handling moving query patterns. Our method does not suffer from the problem of fork dilemma because it is not calculated as a linear function of time alone. By contrast, in our method we use the time and the concept of the safe region together, hence, if the query makes a sudden turn, the result will not be affected because the query will still be located inside the safe region.

Our technique of monitoring the moving query was published in (Al-Khalidi et al., 2014, 2013a).

### 1.3.4 Lookforward Moving Range Query

In our thesis, we focus on a special type of query called moving range query. All traditional range queries try to find all objects of interest around the query point without direction constraint. We propose a novel query processing technique for moving range query based on spatial network databases, called lookforward moving range query (*LMR*). In this technique, we introduce a new way to distinguish between the significant important objects and the minor important objects inside the boundary of the moving range query.

We only take attention of the objects that are located in the direction of the user while he/she is moving, and exclude all objects that are behind the moving user or will make him/her divert away from his/her trajectory. We also improve the selectivity of the filter step to reduce the number of the candidate objects, and consequently, minimise the number of communications between the mobile device and the database server. The resulting technique achieves a better running

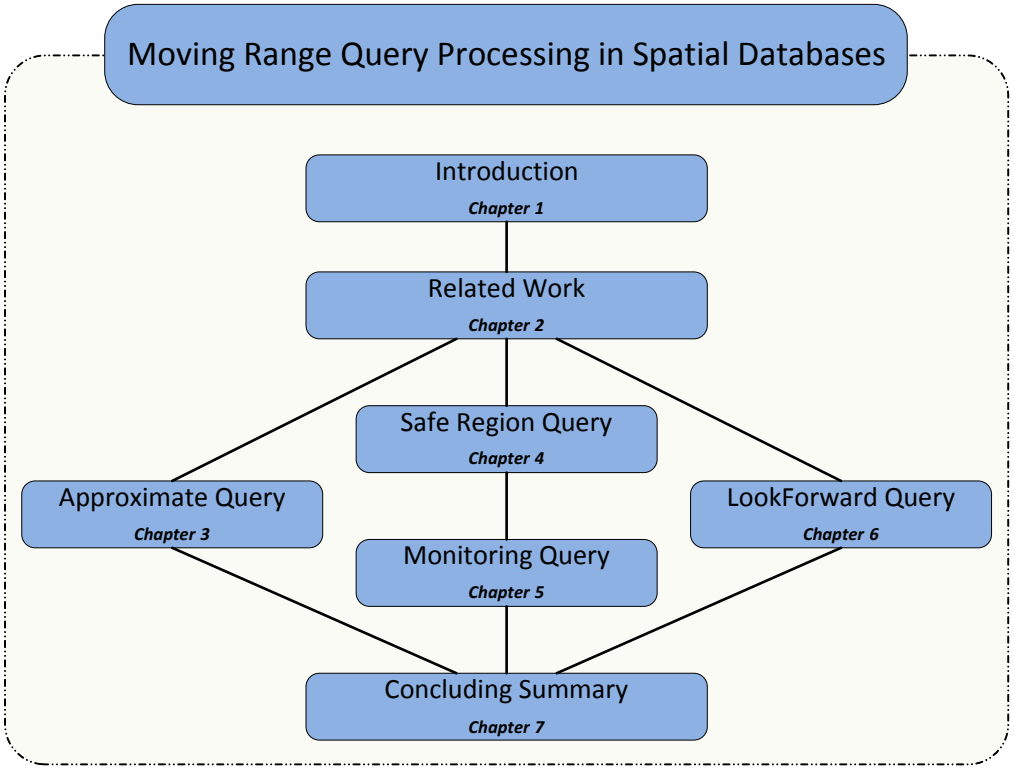



Figure 1.2: Study organisation

time and delivers a better performance, having low a member of split points. To the best of our knowledge, this is the first work dealing with the efficient processing of exclusion of some objects inside the boundary of the range query which are unimportant to the user. We have employed the advantages of the approximate trajectory to achieve a better result for forward objects of interest.

## 1.4 Thesis Organisation

This thesis is organised, as referred in Figure 1.2, as follows:

- Chapter 2 reviews the relevant existing work and basic theoretical background of different spatial query types in spatial and temporal databases.

- Chapter 3 presents novel approximate queries for static and moving queries based on Euclidean and road network databases.
- Chapter 4 presents new types of dynamic safe regions for moving range queries. The size of these safe regions varies depending on the objects' locations around the query.
- Chapter 5 presents a new method to monitor the query (client) itself in order to divide the load between the server and the client and to give the query the ability to move randomly. This method overrides the problem of fork dilemma by employing the linear function and the safe region together.
- Chapter 6 presents a novel lookforward moving range query based on road networks, to emphasis the objects are located in the direction of the moving query, and also to limit the query from diverting back from its original path.
- Chapter 7 concludes this thesis and discusses directions for future work.

# Chapter 2

## Related Work

# 2

# Related Work

In this chapter, we will present the existing related work in order to gain a sufficient understanding of some of the key elements in the spatial queries field. We start by describing range query processing in mobile navigation applications in Section 2.1. Section 2.2, presents the related work of range queries based on road networks. Section 2.3, describes the related work for approximate queries, and in Section 2.4 we provide an overview of the related work for safe region. Finally, in Section 6, a summary of the existing limitations will be presented.

## 2.1 Range Query Processing based on Euclidean Distance

Range query is one of the most frequently used queries in Geographical Information Systems (GIS), such as *Google Maps*, *Whereis Maps*, *Bing Maps* and mobile devices, and also in other areas such as multimedia database queries (Bustos & Navarro, 2009; Waluyo et al., 2004). The range query is applied to find all objects of interest within a given region or radius, and it can be applied

whether the users are moving or not. A range query can be either a static range query or a moving range query. Each is now discussed in turn.

### 2.1.1 Static Range Query

Static range query (or for simplicity range query) is applied when the user is not moving and demands a set of objects of interest. Range query depends on the current location of the user, and it can be defined as: given a query point $q$ (user's location or query location), a radius $e$ (the range of the search specified by the user) and a set of special objects $P$ (e.g., hospitals or restaurants), find all objects of interest $P$ within radius $e$ from $q$.

Most literature regarding range approaches is based on R-tree (Guttman, 1984), which is used to index multidimensional information due to its efficiency and good performance. Also these approaches employ the branch-and-bound searching algorithm (Roussopoulos et al., 1995) to query spatial points storing in R-tree. Figure 2.1 illustrates an example of a range query in a digital map. The objects of interest (i.e., restaurants) are listed by the numbers 1 to 13. The user wants to have all objects within $2km$ from where she stands. The highlighted objects with red colour represent objects of interest that will be received by the user, and the highlighted objects with blue colour represent objects out of the range that will not be included in the result.

Consider Figure 2.2 which shows the processing of range query using R-tree. The range usually corresponds to a circular area or a rectangle window around a query point which is the centre of the range. All objects whose location fall within the range area will be objects of interest. There are three entries for each node, where the objects $\{a,b,...,k\}$ represent the set of points. The traversal on R-tree starts from the root in a depth first manner, visiting only the internal

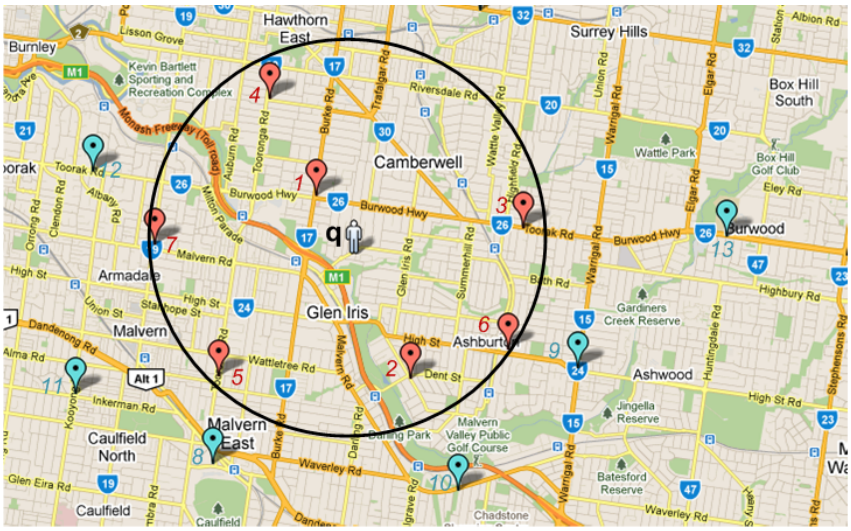



Figure 2.1: An example of range query in mobile navigation

nodes that have minimum distance (*MinDist*) to the query point, which is equal or less than $e$ (e.g, $R1$, $R2$). This process is recursively repeated until all of the leaf nodes that have equal or less minimum distance than $e$ from $q$ are detected. In this case, non-intersecting entries are pruned because they do not have a qualified point (e.g., $R3$, $R7$), and also because their minimum distance from $q$ is greater than $e$. The step of pruning non-qualified nodes is called *filter step*, and the output from filter step should pass a *refinement step* since the object should be examined to specify the result.

In general, range query requires filter step and refinement step (Papadias et al., 2003; Philippe Rigaux, 2002; Safar, 2005) to obtain the essential result. Filter step is used to select objects whose MBR (minimum boundary rectangle) overlaps with the range query to obtain candidates that fall within a specific range $e$, pruning all MBRs with *MinDist* to $q$ that are greater than $e$. Refinement step, on the other hand, is used to sequentially scan the objects that pass the filter step, and then perform the special test on the actual geometry of the objects whose MBRs satisfy the filter step.

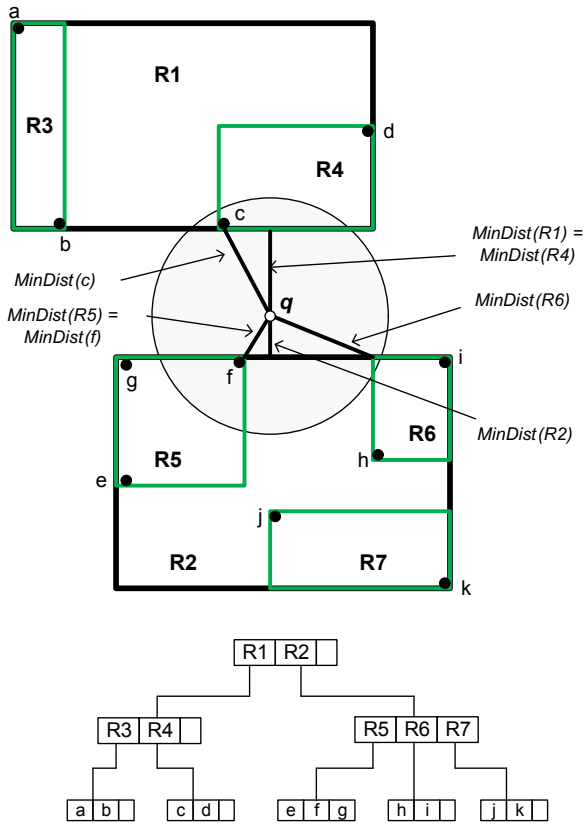



Figure 2.2: Range query using R-tree

The refinement step is done *on-line* (Papadias et al., 2003; Safar, 2005). The objects that pass the filter step successfully, but fail to pass the refinement step, are called *false hits*. Consequently, lots of false hits will lead to extra time to search and an enormous amount of communication between the mobile device and the database server. Also, false hits cause extra input/output access, where input/output access is considered to be the most expensive operation in computer systems, as it requires disk arm movement, which is very slow in comparison to central processing unit (CPU) operations. Therefore, minimising input/output access is the ultimate objective in any query processing algorithms (Taniar et al., 2008).

Several studies have been conducted during the last decade to process range queries efficiently and accurately (Papadias et al., 2003; Mokbel et al., 2004,

2005; R. Cheng et al., 2007; Stojanovic et al., 2008; Pesti et al., 2010; Wang & Zimmermann, 2011). Thus, there is a demand for further improvements to be made for a number of situations. First, the excessive amount of time that may be needed to obtain the results. Second, many false hits are retrieved when the radius $e$ is expanded, which makes the range query processing time consuming. Finally, if the density of the objects is very high, the performance of the range technique will dramatically decrease. However, the main disadvantage of range query is the retrieval of many redundant false hits.

### 2.1.2 Moving Range Query

Range query is location-dependent, which is based on the current users location (Bustos & Navarro, 2009; Xuan, Zhao, Taniar, Safar, & Srinivasan, 2011; Cheema et al., 2011, 2013). The results are based on a static range query, and once the results are assembled by the server then they will be sent to the query user. The static range query is not adequate for moving users. In the moving range query, also called continuous range query, users may use a predefined path or move randomly. The moving range query can be considered as two cases: predefined or random moving range query. They are now discussed.

#### Moving Range Query - Predefined Path

In the predefined path, moving range query can be defined as: given a set of special objects $P$ (e.g., hospitals), and a query path $q = [S,D]$ (user's path between two points, $S$ = start point and $D$ = destination (end) point), and radius $e$, retrieve all objects within the distance $e$ to every point in the query path $q$ (line segment).

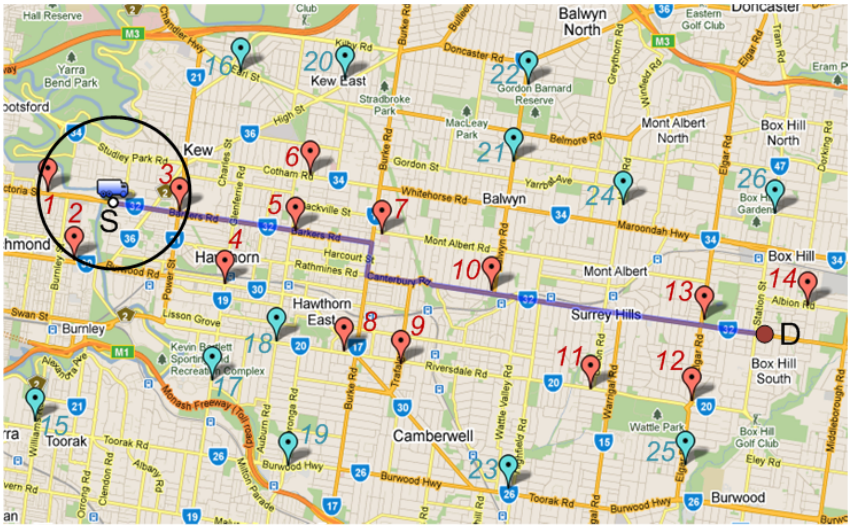



Figure 2.3: An example of moving range query in mobile navigation

Figure 2.3 illustrates an example of moving range in a digital map. The objects of interest (i.e., restaurants) are listed by the numbers 1 to 26. The user wants all objects within 1$km$ while she is driving (moving) from the start point ($S$) to the destination point ($D$). All of the highlighted objects in red will be in the result list, while the highlighted objects in blue will not be included in the result list.

The problem of dealing with data that are continuously changing, due to the position of moving objects, was first addressed by Sistla et al. (Sistla et al., 1997). They identified the importance of the continuous concept in a nearest neighbours query, and therefore, they proposed a new data model to represent moving objects in data systems. The user's position was represented as a function of time, that is, it changes as time passes, even without an explicit update. Sistla et al. proposed a query language which enables the specification of future queries (i.e., queries that refer to future states of the database). However, they did not discuss access or processing methods (Tao et al., 2002).

In another study, Song et al. (Song & Roussopoulos, 2001) proposed the first algorithm for continuous nearest neighbour query processing. This algorithm employed sampling to compute the result, but it suffered from the common drawbacks of sampling. Therefore, if the sampling rate is low, then the result will be incorrect, otherwise, significant computational processing is required. Tao et al. (Tao et al., 2002) proposed a solution based on performing one single query for the entire path. They extend the approach to address continuous $k$ nearest neighbours queries. Most of the efforts, afterwards, were concentrating on improving the performance of continuous $k$ nearest neighbours processing methods (Safar & Ebrahimi, 2006; Zhao et al., 2014).

The aim of the moving query is to find split points (split nodes) (i.e., the locations where the query results will be updated) to reduce the communication cost of tracking the moving query. Also, the server needs to inform the user when the range query results change. Consequently, a new need for continuous online communication between the user and the database server appears due to the necessity of updating. It is also necessary to mention that in wireless environments, mobile devices suffer from limited bandwidth and low-quality communications (Chow et al., 2009). However, the following problems occur:

1. The number of split points is twice as many as the number of objects of interest (Xuan, Zhao, Taniar, Rahayu, et al., 2011). This leads to a significant increase in communication between the mobile user and the server.

2. Fast updating of the results in areas with a high density of objects deprives the user of the opportunity to make a decision.

3. The existence of critical objects (i.e., objects with minimum distance to the query path equal or almost equal to $e$, $D_E(q, p_i) \approx e$) that enter the

boundary of the query for a very short period poses two problems: *i*) *false misses*, where relevant objects are not included in the result, and *ii*) when critical objects are present only for a short time period, the user then has insufficient time to plan and react.

The main disadvantage of moving range query approaches is the increased number of updating results that lead to a raise the number of communications with the server.

**Moving Range Query - Random Path**

To track a moving query in a non-defined path (or random path), the concept of path prediction was introduced. Path prediction enables better results and reduces the location update frequency in object tracking while preserving accuracy. Different models for predicting the future position of a moving query have been proposed over the years; however, they only offer accurate route predictions in the short term.

A simple prediction model represents an object's future location by a *linear function of time*, based on the most recently reported location and velocity of the object (Jeung et al., 2010). This representation is typically adopted in the context of indexing (Jensen et al., 2004; Šaltenis et al., 2000) because it is compact, easy to obtain, and reduces the amount of updates compared with constant functions of time. However, this model does not offer accurate predictions beyond the short term. Furthermore, linear predictions suffer from the *fork dilemma* for path prediction. Therefore, a more complex, non-linear prediction model has been introduced. In this model the *recursive motion function* (Tao et al., 2004) achieves better predictions by finding a curve that best fits the last few reported locations of a moving object. However, in this model, the problem of fork

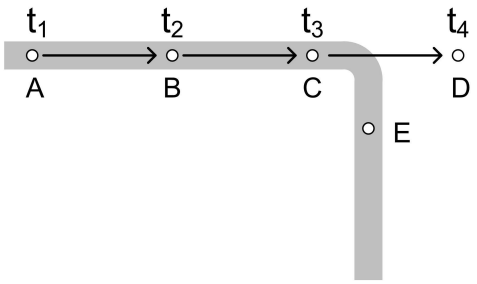



Figure 2.4: Fork dilemma when predicting a moving query

dilemma has not been solved, and the model fails to predict sudden direction changes (i.e., turns). Figure 2.4 shows the fork dilemma problem that linear predictions are suffering from during path prediction. Linear prediction fails to predict the object's movement at the turn.

Generally, inaccurate prediction of a random moving query leads to frequent updates of the object's location (i.e., communication is an expensive operation). Therefore, reducing the number of location updates is necessary since users do not need to inform the server of their location as long as they follow the predicted path, which is known to both the server and the client.

## 2.2 Range Query Processing based on Road Network Distance

Range query processing depends on Euclidean distance to provide the relative position of the spatial object. In some circumstances, the location of spatial objects may need to be specified by the underlying network and not by Euclidean space. Thus, researchers (Papadias et al., 2003; Safar, 2008; Ghadiri et al., 2011; Xuan, Zhao, Taniar, Safar, & Srinivasan, 2011) investigated range query using network distance. The aim of this query is to find spatiotemporal object(s) of interest which satisfy expected spatiotemporal relationships given segments of

trajectory(ies). This type of query is to retrieve all objects within the diameter of the query ($q$) in a trajectory segment(s). For example, find all post offices along the specific trajectory segment from $q$ to distance $e$.

In 2003, Papadias et al. (Papadias et al., 2003) proposed two algorithms, *Range Euclidean Restriction (RER)* and *Range Network Expansion (RNE)*, taking advantage of location and connectivity to efficiently prune the search space. RER and RNE can find all objects of interest within the given region or radius, and can be defined as: given $q$ a query point (user's location), $e$ a radius (the range of the search specified by the user) and $P$ a set of special objects (e.g., post office or petrol station), find all object(s) of interest $pi \in P$ within network distance $D_N$ from $q$.

To give an example of a range query in a GPS map, consider Figure 2.5 where objects of interest (i.e., takeaway shops) are listed by the numbers 1 to 17. The user is asking for all of the objects within $2.5km$ from where he stands. The red objects (i.e., 1-6) represent objects of interest that will be received by the user because their network distance is less than $2.5km$. The blue objects (i.e., 7-17) represent objects out of the range where their distance to $q$ is more than $2.5km$, consequently, this will exclude them from the result list.

RER and RNE algorithms give a solution for range queries in network databases by introducing an architecture that integrates the network and Euclidean information and captures spatial entities. However, the number of communications is fairly high when performing these two methods in addition to their costly pre-computations. Hence, using such methods in real life applications is not reliable.

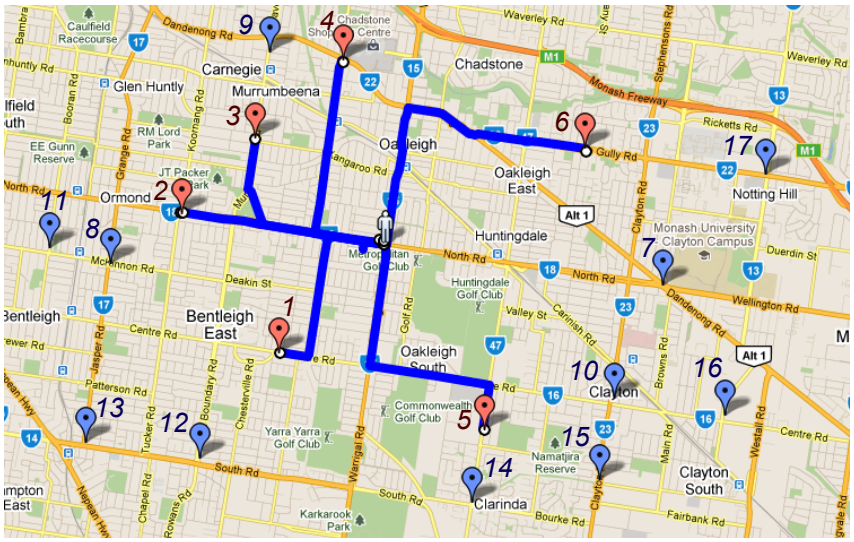



Figure 2.5: Range query in a spatial network database using an online map

### 2.2.1 Range Euclidean Restriction

The RER algorithm is based on a Dijkstra algorithm to find the shortest path. By using R-tree, the RER method first performs a range query at the entity dataset and returns the set of qualified objects $S'$ within the diameter of the range which is based on the Euclidean distance from the query ($q$). This range diameter (Euclidean distance) serves as the upper boundary of the search area and then retrieves the data entities falling on these segments. $S'$ is guaranteed to avoid false misses (i.e., $D_N(q,p) \leq D_E(q,p) \leq diam/2$, where $D_N$, $D_E$ and $diam/2$ refer to the network, Euclidean distance and radius of the query, respectively), but it may contain a large number of false hits. However, the set of objects $S'$ have to be refined to filter the false hits from the candidate objects by applying network expansion in the next step until all the candidates are tested or all the segments in the range are exhausted.

### 2.2.2 Range Network Expansion

The RNE algorithm first computes the set *QS* of qualifying segments (paths branching out from the query point) within network distance $e$ from $q$ and then retrieves the data entities falling on these segments. In other words, set all paths (segments) that may contain object(s) of interest in *QS*, and then evaluate these paths in *QS* to find the objects of interest. This is done by traversing the index tree, such as R-tree, and while visiting each node in the index tree, joining the node of the index tree with the qualifying segments *QS*, in order to check whether the node (e.g., MBR if it is a non-leaf node, or object of interest if it is a leaf node) is located in *QS* or not.

## 2.3 Approximate Query Processing

The exhaustive processing to find the exact answer to a query can be prohibitively expensive, depending on the query nature and data properties. This is generally due to the exponential nature of the problem search. The relatively high complexity of range searching has led researchers to consider the problem in the context of approximation. A geometric way to do this is to consider the range shape to be fuzzy, and allow points that are close to the range boundary to either be counted or not. The user supply approximation parameter $\varepsilon > 0$ (i.e., $\varepsilon$ distance absolute error), where the range shape $e$ is bounded and points lying within distance $\varepsilon \, . \, diam(e)$ of the boundary of the range may or may not be included (da Fonseca & Mount, 2010).

Many researchers have proposed improvements in the performance of the approximate search, however most of their work has been related to the performance of the approximate nearest neighbour search query in Euclidean space

and its drawbacks. Bern (Bern, 1993) considered the problem of the approximate nearest neighbour (i.e., closest-point problem). He proposed a data structure based on quadtrees, and used Euclidean space which provided logarithmic query time. However, the approximation error factor for his algorithm was a fixed function of dimension. Furthermore, Arya and Mount (Arya & Mount, 1993) proposed a data structure which achieves poly-logarithmic query time in the expected case, and nearly linear space. In their algorithm the approximation error factor was an arbitrary positive constant, fixed at preprocessing time. This algorithm found the nearest neighbour in a moderately large dimension significantly faster than other existing practical approaches.

In another study, Arya et al. (Arya et al., 1998) strengthened the results of that algorithm significantly by proposing a method called $\varepsilon$-approximate method which guaranteed that the quality of the result is bounded by some constant in terms of distance error, given a positive tolerate real $\varepsilon$ $(\varepsilon \geq 0)$ as a maximum distance relative error. However, in the $\varepsilon$-approximate method, the trade-off between the cost and the accuracy of the result cannot be controlled easily since it depends on the value of $\varepsilon$ which is an unbounded positive real. Corral et al. (Corral et al., 2002; Corral & Vassilakopoulos, 2005), on the other hand, proposed a new version of approximate nearest neighbour called $\alpha$-allowance method to bound the distance relative error $\gamma$.

In a different study, Chow et al. (Chow et al., 2009) proposed an algorithm of approximate range nearest queries, to return an answer set that includes the approximate nearest object(s) to every point in a range area. Their algorithm minimised the number of objects (i.e., returned as objects of interest ) in order to minimise the transmission time of sending the answer to the user. They used a Voronoi diagram to index the objects which partition the space into a set of polygons so that each polygon has exactly one object. Another study

was conducted by Arya et al. (Arya et al., 2012, 2009), which modified the index structure of the objects to get the result of the range search. The result is approximate depending on the relative error that has been given by the user. Subsequently, Fonseca et al. (da Fonseca et al., 2013; da Fonseca & Mount, 2010) proposed a similar data structure to retrieve the approximate result but it depended on the absolute error.

## 2.4 Safe Region Processing

Recent developments in mobile communication technologies have brought dramatic and fundamental changes to our everyday lives. This is due to the growing demand for a technology to help users navigate crowded roads while they are moving, guiding them to the best route, and answering their queries (Stojanovic et al., 2008; Ogiela & Ogiela, 2009). Mobile navigation and location aware systems are two applications among the most prominent mobile information services (Ilarri et al., 2012). Many different types of moving queries have been used in such applications.

A moving query continuously returns sets of objects of interest which extend from the registration of the query to its cancellation. Over this time, the query results must continuously be updated even when the query conditions remain unaltered. Therefore, to reduce the updating costs when a query is continuously moving the safe region has been proposed, where the set of the objects of interest does not change as long as the moving query remains in this region.

Assume in Figure 2.6 that a user wants to know the closest two fuel stations as he/she moves along. In this scenario, the user (at position $q$) poses a query to the server returning points $p1$ and $p2$. After processing the query, the server returns the result to the user. The reason behind introducing the safe region is

to keep this result the same as long as the user remains in a certain area around the initial position, which we are referring to using a shaded area. In addition to the query result, the server has to return the dimension of the query's safe region, according to which the user can determine whether a new query should be issued or not by checking whether it is still inside the safe region. Given the fact that moving users have limited storage and limited processing capabilities compared to the server, it put the burden on the server side to perform additional processing for the initial query in order to reduce the number of the subsequent queries. Moreover, the representation of the safe region has to be compact in order to reduce the network cost, the storage requirements and the computation that must be performed at the user side.

Many safe region methods have been proposed to achieve efficient evaluation by reducing the communication and updating costs. Some of these methods apply time-based techniques (R. Cheng et al., 2007; Hu, 2005), in which users need to report their query locations to the server after every $t$ time units to attain up-to-date information. While others employ distance-based techniques (Zhang et al., 2003; Cho et al., 2013), in which users need to report their query locations to the server after every $d$ distance units. However, neither time-based nor distance-based techniques are accurate in obtaining a correct query response when they use a fixed time or a fixed distance frame for constructing the safe region. Therefore, some distance-based techniques have used a dynamic distance for constructing the safe region, which is accurate in obtaining the response, however, these techniques do not predict when and where the mobile user will leave the safe region.

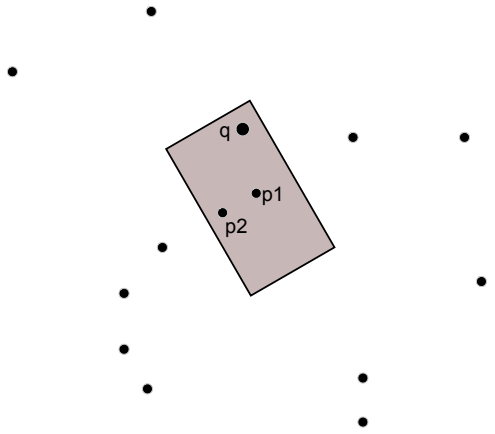



Figure 2.6: Safe region

### 2.4.1 Fixed Safe Region

Hu et al. (Hu, 2005) proposed a generic framework to handle continuous queries with safe regions whereby the location updates from mobile clients are further reduced. This study was limited in that it was based on the assumptions that queries are static which might not be feasible in real-world applications. Another approach proposed by Cheng et al. (R. Cheng et al., 2007), was a time-based location update mechanism which was designed to improve the temporal data consistency for the objects relevant to queries. In this type of safe region, the query sends an update containing its current location to the server every timestamp or fixed distance. To be able to send location updates more frequently, data objects with significance to the correctness of query results are required. However, this method had a main limitation: an object would repeatedly send location updates to the server when it is enclosed by a query region, that does not need it (Hsueh et al., 2009).

Table 2.1: Summary of query types and their limitations

| Query | Limitations |
|---|---|
| Static range query | • Time consuming<br>• Many false hits<br>• Expensive computations<br>• Efficiency is inversely proportional to the density |
| Moving range query (path defined) | • Many split points<br>• Fast updating to the results<br>• Existence of critical objects<br>• Continuous online communications<br>• The user has no privacy |
| Random moving range query (path prediction) | • Inaccurate predictions for long term<br>• Fork dilemma<br>• Frequent updates to the query's location<br>• Continuous monitoring |

### 2.4.2 Dynamic Safe Region

In 2003 researchers proposed a dynamic safe region (Zhang et al., 2003). Its main limitation is that the existing approaches use rectangular safe regions only, and they are not applicable to the moving circular range queries. A recent search (Yung et al., 2012; Cheema et al., 2013, 2011) introduced a circular safe region for a moving query. In this type of safe region, the query has to update frequently enough to keep a reasonable tracking precision. This method did not calculate the area, which means the server is always in a standby situation expecting contact from the query. In addition, this method did not show how the server will be aware of the new location of the user when the user leaves its current safe region.

## 2.5 Summary

In this chapter, we have shown that most studies on range queries attempted to process static and moving range queries efficiently and accurately. Nevertheless, there is a demand for further improvements to be made to address limitations summarised in Table 2.1. We also discussed the approximate concept in spatial databases where we examined the work of many researchers, however their work related to the performance of the approximate nearest neighbour query in Euclidean space. Then we explored the principle of the safe region in location-aware mobile devices and we reviewed some safe region techniques in moving queries.

# Chapter 3

# Approximate Static and Moving Range Query



**Publications and Submissions:**

1. AL-Khalidi, H. Taniar, D. and Safar, M. (2013), Approximate algorithms for static and moving range queries in mobile navigation. In *Computing*. Springer, 95(10-11), pp.949-976.

2. AL-Khalidi, H. Abbas, Z. and Safar, M. (2013), Approximate range query processing in spatial network databases. In *Multimedia Systems*. Springer, 19(2), pp.151-161.

# 3

# Approximate Static and Moving Range Query

A spatial range query is used to find objects of interest within a given radius. This type of query is common in a mobile environment for mobile users to be able to find surrounding objects of interest. Traditionally, a range query will return all objects within a given radius. However, in many circumstances, having all objects is not necessary, especially when there are already enough objects close to the query point. In other circumstances, expanding the radius may give users better results, especially when there are a lot of objects just outside the search boundary. Therefore, in this chapter, we present new range queries, called approximate static range query and approximate moving range query, where the query results are approximate, rather than exact.

The rest of the chapter is organised as follows. Section 3.1 states our motivation, followed by the next two Sections 3.2 and 3.3 in which we introduce our proposed methods based on Euclidean distance, that is, approximate static range query and approximate moving range query, respectively. Section 3.4 and Section 3.5 present our methods: approximate static range and approximate

moving range query, based on road network distance. Performance experiments are covered in Section 3.6. Finally, Section 3.7 summaries the chapter.

## 3.1 Motivation

Recent developments in mobile communication technologies have brought dramatic and fundamental changes in our everyday life. One of the most noted and growing applications of mobile information services is mobile navigation, due to the growing demand for the technology that helps users navigate on crowded roads, guides them to the best route, and gives answers to their queries (Zhao et al., 2014).

The high cost of executing users queries has led to the quest for query search techniques that can be performed efficiently. Therefore, the context of approximation has been considered (Arya et al., 2009). In practice, not all applications demand exactness in the location information of the object of interest (Petkova et al., 2009). For example, a passenger wants to know if the next bus is arriving soon, or a library member wants to know if a specific book has been returned. Hence, the exact location is not necessary.

On the other hand, in applications such as Geographic Information Systems (GIS) and cellular networks, the amount of data can be very large and a huge number of objects of interest for the final result may need to be considered. Also, it is important to consider the huge cost with respect to response time and input/output. Consequently, the use of the approximate techniques are worthy of examination, especially when the results are obtained at much lower cost (Corral et al., 2002).

In this chapter, we are going to improve the processing of range queries (i.e., static and moving) using approximation. We will introduce novel techniques to

process such types of queries efficiently. We show that approximate results are as valuable as exact results, however, they can be obtained much faster than the exact results with less cost. To the best of our knowledge, no studies have been conducted in approximate static range query relying on a distance based query. Moreover, there is no work that deals with the moving range query in the context of approximation. Our research presented in this chapter was published in (Al-Khalidi et al., 2011; AL-Khalidi et al., 2013; AL-Khalidi, Taniar, & Safar, 2013).

## 3.2 Approximate Static Range Query based on Euclidean Distance

The consideration of search time is important in spatial databases, and in the state of the range query, the search time is affected by the number of retrieved objects and the size of range query $e$.

In general, the range query retrieves all objects that are within a certain range $e$ from a given query point $q$, thereby necessitating an excessive amount of time to obtain the essential results. Also, when the search is finalised and returns a result of either a massive number of objects or absolutely no results, then users need to submit another query which leads to another search. Furthermore, repeated false hits (i.e., irrelevant objects as candidates) that occur during the search should be considered since each false hit is a waste of search time. Thus, we consider these problems in the context of approximation, since approximate results are acceptable, and they can be obtained much faster. On the other hand, $K$NN ($K$ nearest neighbour) is not suitable to solve all these problems, because

this method retrieves a specific number of objects ($K$) regardless of the distance from the query.

We propose three novel approximate range methods for static query, namely *Lowerbound approximate static range query* (*Lowerbound* ASR) and *Upperbound approximate static range query* (*Upperbound* ASR) and *approximate static range query* (ASR) to enhance the performance of the range query by using different factors to bound the query. The following subsections will describe these methods.

### 3.2.1 Lowerbound Approximate Static Range Query (*Lowerbound* ASR)

The *Lowerbound* ASR method reduces the search time by retrieving fewer results. *lowerbound* $e \times (1-\gamma)$ is introduced where all of objects of interest within the *lowerbound* range will be retrieved, in addition to retrieving some of the objects of interest that fall between the *lowerbound* range and the exact range. There are two reasons to retrieve less results. The first is to avoid retrieving objects of interest that fall away from the query $q$ (especially in a high density environment). The second is to return the result in a short time.

The *Lowerbound* ASR functions as follows: Given a tolerated positive real $\gamma$ $(0 \leq \gamma \leq 1)$ as maximum distance relative error, the result of *Lowerbound* approximate static range query (*Lowerbound* ASR) within distance $e$ is $(1-\gamma)$ approximate range. When the pruning heuristic is applied on the *Lowerbound* ASR algorithm, the following will take place:

1. Filter step: MBR $R$ will be discarded, if $MinDist(R,q) > e \times (1-\gamma)$.

2. Refinement step: an object $pi$ is discarded, if $MinDist(pi,q) \geq e$.

The decrement of the *lowerbound* depends on the value of $e$ multiplied by $(1-\gamma)$. If $\gamma = 0$, then the approximate algorithm behaves just as the exact range query algorithm, where the output represents the exact results. Also, if $\gamma = 1$, then the approximate algorithm will retrieve all the objects within distance $e$ from $q$ and these objects are within the MBRs that intersect $q$.

Let $P$ be a point dataset $(P \neq \phi)$ in d-dimensional Euclidean space $P \subset E^d$, $e$ the radius of the query $(e \in \Re^+)$ and $q$ query point $(q \in E^d)$. Then, the result of the *Lowerbound* approximate static range query with respect to a query point $q$ is the set *Lowerbound* $ASR(P', q, e, -\gamma)$. To find the *Lowerbound* ASR, two parts of the result should be found:

1. The exact range search *RS*, the exact result, to the area that is within the lowerbound $(P, q, e \times (1-\gamma)) = \{\ (p1, p2, ..., pm) \in P^m : pi \neq pj\ \ i \neq j\ \ 1 \leq i, j \leq m$ and

   $\forall i \in P - \{p1, p2, ..., pm\}, dist(p1, q) \leq dist(p2, q) \leq ... \leq dist(pm, q) \leq e \times (1-\gamma) \leq dist(pi, q)\}$

2. The $(1-\gamma)$approximate range $AR^-$, the approximate result, to the area that is between the *lowerbound* and the exact range $(P'', q, e, -\gamma) = \{(p'1, p'2, ..., p'z) \in P^z : p'i \neq p'j\ ;\ i \neq j\ ;\ m \leq i, j \leq z$ and

   $\exists p'i \in P - \{p1, p2, ..., pn, p'1, p'2, ..., p'z\}, e \times (1-\gamma) \leq dist(p'1, q) \leq dist(p'2, q) \leq ... \leq dist(p'z, q) \leq e \leq dist(p'i, q)\}$

In other words, *Lowerbound* $ASR(P', q, e, -\gamma) = RS(P, q, e \times (1-\gamma)) \cup AR^-(P'', q, e,\ -\gamma))$

In *Lowerbound* ASR, we use R-tree to answer the query starting from the root. Each MBR that intersects with the *lowerbound* is recursively searched looking for the qualifying points and in the same time the non-intersecting

MBRs are pruned. Up to this stage, R-tree just provides the filter step. The output of the filter step has to pass the refinement step, which in turn examines the objects to achieve the result. Any object *pi* that passes the filter step and $MinDist(pi,q) > e$ will be excluded from the result. Our approach, *Lowerbound* ASR, guarantees that any object *pi* that passes the formula $(MinDist(pi,q) - e)/e \leq -\gamma$ is included in the result. In this case, the algorithm discards all MBRs $R$ when $MinDist(R)$ is larger than $e \times (1-\gamma)$, and the objects not within those MBRs and away from $q$ by distance $e$ are also discarded.

### 3.2.2 Upperbound Approximate Static Range Query (*Upperbound* ASR)

*Upperbound* ASR gives alternative choices to the user since more objects are retrieved without extra search time. This approach guarantees the retrieval of all objects of interest within $e$ distance from $q$ (i.e., the exact result) in addition to retrieving some of the objects of interest that fall just out of the range distance $e$, but within a distance relative error $(\gamma)$. There are two reasons for retrieving this extra result: first, to avoid the null result (especially in a low density environment) and second, to give the user more choices without affecting the search time and also to avoid having another search.

*Upperbound* ASR functions as follows: Given a tolerated positive real $\gamma$ $(0 \leq \gamma \leq 1)$ as maximum distance relative error, the result of the *Upperbound* approximate static range query (*Upperbound* ASR) within distance $e$ is $(1+\gamma)$ approximate range. When the pruning heuristic is applied on the *Upperbound* ASR algorithm, the following occurs:

1. Filter step: MBR $R$ will be discarded, if $MinDist(R,q) > e$ .

2. Refinement step: an object *pi* is discarded, if $MinDist(pi,q) > e \times (1+\gamma)$.

The increment of *upperbound* depends on the value of $e$ multiplied by $(1+\gamma)$. If $\gamma = 0$, then the approximate algorithm behaves as an exact range query algorithm, where the output represents the exact precise solution. Also, if $\gamma = 1$, then the approximate algorithm will retrieve all objects within distance $2e$ from $q$, and these objects are within the MBRs that intersect the range search $e$ (i.e., the boundary of the query).

Let $P$ be a point dataset $(P \neq \phi)$ in $E^d$, and $e$ the radius of the query $(e \in \Re^+)$ and $q \in E^d$. Then, the result of the *Upperbound* approximate static range query with respect to a query point $q$ is the set *Upperbound* $ASR(P', q, e, +\gamma)$. To find the *Upperbound* ASR, two parts of the result should be found:

1. The exact range search $RS(P,q,e) = \{(p1, p2, ..., pn) \in P^n : pi \neq pj \ ; \ i \neq j \ ; \ 1 \leq i, j \leq n$ and

   $\forall pi \in P - \{p1, p2, ..., pn\}, \ dist(p1,q) \leq dist(p2,q) \leq ... \leq dist(pn,q) \leq e \leq dist(pi,q)\}$

2. The $(1+\gamma)$approximate range $AR^+(P'', q, e \times (1+\gamma)) = \{(p'1, p'2, ..., p'z) \in P^z : p'i \neq p'j \ i \neq j \ n \leq i, j \leq z$ and

   $\exists p'i \in P - \{p1, p2, ..., pn, p'1, p'2, ..., p'z\}, \ e \leq dist(p'1,q) \leq dist(p'2,q) \leq ... \leq dist(p'z,q) \ \leq e \times (1+\gamma) \leq dist(p'i,q)\}$

In other words, *Upperbound* $ASR(P', q, e, +\gamma) = RS(P,q,e) \cup AR^+(P'', q, e, 1+\gamma))$

In *Upperbound* ASR, we use the R-tree to answer the query starting from the root. Each MBR that intersects with the range query $e$ is recursively searched looking for the qualifying points, and non-intersecting MBRs are pruned. Up to this stage, the R-tree just provides the filter step. The output from the filter step has to pass the refinement step, which in turn examines the objects in order to

produce the result. Any object $pi$ that passes the filter step and $MinDist(pi,q) > upperbound$ will be excluded from the result. Our *Upperbound* ASR approach guarantees that any object $pi$ that passes the formula $(MinDist(pi,q)-e)/e \leq \gamma$ is included in the result. In this case, the algorithm discards all MBRs $R$ when $MinDist(R,q)$ is larger than $e$, and the objects not within those MBRs and away from $q$ by distance *upperbound* are also discarded.

### 3.2.3 Approximate Static Range Query (ASR)

The ASR method combines *Lowerbound* ASR and *Upperbound* ASR approaches to give the result in a short time, and also to give the user extra choices without affecting the search time and to avoid another search.

ASR functions as follows: Given a tolerated positive real $\gamma$ $(0 \leq \gamma \leq 1)$ as maximum distance relative error, the result of approximate static range query ASR within distance $e$ is $(1 \pm \gamma)$ approximate range. When the pruning heuristic is applied in the ASR algorithm, the following occurs:

1. Filter step: MBR $R$ is discarded if $MinDist(R,q) > e \times (1-\gamma) \Longleftrightarrow MinDist\ (R,q) > lowerbound$.

2. Refinement step: an object $pi$ is discarded if $MinDist(pi,q) > e \times (1+\gamma) \Longleftrightarrow MinDist(pi,q) > upperbound$.

Let $P$ be a point dataset $(P \neq \phi)$ in $E^d$, $e$ the radius of the query $(e \in \Re^+)$ and $q \in E^d$. Then, the result of the $\gamma$ approximate method with respect to a query point $q$ is the set $ASR(P,q,e,\gamma)$.

To find the approximate range query, two parts of the result should be found:

1. The exact range search for the *lowerbound* $RS(P,q,e \times (1-\gamma))$=$\{(p1,p2,...,pn) \in P^n : pi \neq pj\ ;\ i \neq j\ ;\ 1 \leq i,j \leq n$ and

$\forall pi \in P\{p1, p2, ..., pn\}, dist(p1,q) \leq dist(p2,q) \leq ... \leq dist(pn,q) \leq e \times (1-\gamma) \leq dist(pi,q)\}$

2. and the $(1 \pm \gamma)$approximate range $AR^{\pm}$ between the *lowerbound* and the *upperbound* $(P', q, e \times (1+\gamma), \gamma) = \{(p'1, p'2, ..., p'z) \in P^z : p'i \neq p'j \ \ i \neq j \ \ n \leq i, j \leq z$ and

   $\exists p'i \in P - \{p1, p2, ..., pn, p'1, p'2, ..., p'z\}, \ e \times (1-\gamma) \leq dist(p'1,q) \leq dist(p'2,q) \leq ... \leq dist(p'z,q) \leq e \times (1+\gamma)) \leq dist(p'i,q)\}$

On the other hand, $ASR(P,q,e,\gamma) = RS(P,q,e \times (1-\gamma)) \cup AR^{\pm}(P', q, e \times (1+\gamma), \gamma))$

ASR guarantees that any object $pi$ that passes the formula $(MinDist(pi,q) - e)/e \leq -\gamma$ is included in the result. In this case, the algorithm discards all MBRs $R$ when $MinDist(R,q)$ is larger than the *lowerbound* $(e \times (1-\gamma))$ and the objects that are not within those MBRs and away from $q$ by distance *upperbound* $(e \times (1+\gamma))$ are also discarded.

### 3.2.4 ASR Algorithm

R-tree (Guttman, 1984) is used to find objects of interest within the range query $e$. The approximate techniques and branch-and-bound algorithm are applied to prune unnecessary items. In addition, the straightforward search algorithm such as Best-First search is also used to find the objects within a short time (Corral et al., 2002; Corral & Vassilakopoulos, 2005).

The Algorithm 3.1 can find the approximate range based query after giving $q$, $e$ and $\gamma$. $e \times (1+\gamma)$ in the algorithm represents the *upperbound* of the search, and MBRs or objects above this bound are pruned. The *lowerbound* of the search $e \times (1-\gamma)$, and the range of the *lowerbound* represents the exact result

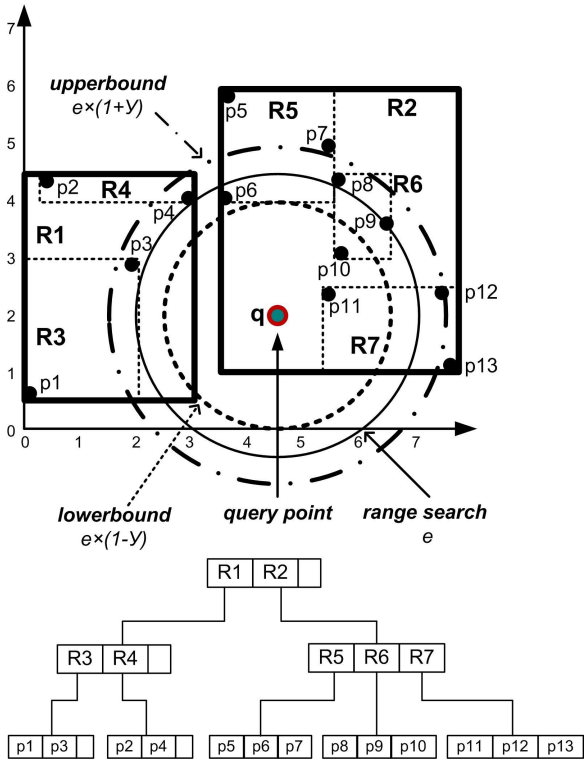



Figure 3.1: Approximate static range query on R-tree

of the search. The range query between the *Lowerbound* and the *Upperbound* represents the approximate static range query, prunes the MBRs and retrieves the objects.

To apply the *approximate static range query* ASR algorithm and compare it with the exact range query consider Figure 3.1 and Table 3.1 (i.e., Table 3.1 represents the minimum distance from the query point $q$ to each single node in Figure 3.1). In Figure 3.1 there are twelve objects $(p1, p2, ..., p12)$, seven intermediate nodes $(R1, R2, ..., R7)$, and a range query $e$ which is equal to 2.5.

In the range query algorithm, the *MinDist* of $R1$ and $R2$ are calculated and en-queued in Q-queue with their *MinDist* 1.5 and 0 respectively. Then the queue will be sorted ascendingly and this will place $R2$ at the top of the queue. In this algorithm, the top of the queue will continually be de-queued and expanded if its $MinDist \leq e$ (i.e. $e = 2.5$). The result of $R2$ expansion

Table 3.1: The minimum distance between item $X$ and the query point $q$

| *MinDist* from $q$ | | | | | |
|---|---|---|---|---|---|
| $R1$ | 1.5 | $p1$ | 4.6 | $p8$ | 2.7 |
| $R2$ | 0 | $p2$ | 4.7 | $p9$ | 2.5 |
| $R3$ | 2.5 | $p3$ | 2.7 | $p10$ | 1.4 |
| $R4$ | 2.5 | $p4$ | 2.5 | $p11$ | 1 |
| $R5$ | 2 | $p5$ | 4.1 | $p12$ | 3 |
| $R6$ | 1.4 | $p6$ | 2.2 | | |
| $R7$ | 0.7 | $p7$ | 3.2 | | |

is $(< R5,2 >, < R6,1.4 >, < R7,0.7 >)$ which will all be en-queued. Then, $R7$ will be de-queued and the outcome of its expansion will be $(< p11,1 >, < p12,3 >)$. As a result, $p11$ will be chosen to join the objects of interest list because its $MinDist \leq e$.

All internal nodes within distance $e$ from $q$ will be checked looking for the object(s) of interest. After checking all the nodes, the final interest objects list will be $(< p11,1 >, < p10,1.4 >, < p6,2.2 >, < p4,2.5 >, < p9,2.5 >)$. In general, the range query algorithm checks all of the seven MBRs and exhausts them to find the result; however, many false hits are generated.

Compared with the range query algorithm, our approximate static range query algorithm operates in a different manner, whereby the numbers of the visited MBRs are reduced; consequently, the number of false hits are reduced too. The above example is used to obtain the approximate result, assuming that the distance relative error is $\gamma = 0.2$.

In this example, the *lowerbound* is 2, $e \times (1-\gamma) = 2.5 \times (1-0.2)$, and the *upperbound* is 3. This means that all objects within Euclidean distance 2 from $q$ should be retrieved, along with some of the objects within Euclidean distance between 2 and 3. $R1$ and $R2$ are en-queued in Q-queue with their *MinDist* 1.5 and 0 respectively. Providing that MBRs that have the smallest distance to $q$ should be visited first, then the result of expanding $R2$ would be

($< R5, 2 >$, $< R6, 1.4 >$, $< R7, 0.7 >$). Next, the result of expanding $R7$ would be: ($< p11, 1 >$, $< p12, 3 >$), and here $p11$ and $p12$ will be added to the interest objects list since their *MinDist* are less than the *upperbound*.

The algorithm will back recursively to expand what is left, and in this case, $R6$ and then $R5$ will be expanded and the result is: ($< p10, 1.4 >$, $< p8, 2.7 >$, $< p9, 2.5 >$) and ($< p6, 2.2 >$, $< p7, 3.2 >$, $< p5, 4.1 >$) respectively. After that, all objects with *MinDist* less than the *upperbound* should be added to the interest objects list; ($p10, p8, p9$ and $p6$). Then, going back recursively to expand $R1$, the result is ($< R3, 2.5 >$, $< R4, 2.5 >$). However, the search is terminated without expanding $R3$ and $R4$ because their *MinDist* are greater than the *lowerbound*.

After executing the approximate static range query, the interest objects result list is: ($< p11, 1 >$, $< p10, 1.4 >$, $< p6, 2.2 >$, $< p8, 2.7 >$, $< p9, 2.8 >$, $< p12, 3 >$); (i.e., three objects within the exact range and three objects within the approximate range). The application of this algorithm reduced the number of the visited MBRs to five compared with seven visited MBRs when applying the range query algorithm. On the other hand, the number of objects that have been checked are eight (i.e., six interest objects and two false hits) compared with twelve objects (i.e., five interest objects and seven false hits) when executing the range query algorithm. Consequently, the execution time needed to obtain the result is reduced with fewer false hits.

In another situation, assuming that we want to retrieve all objects of interest within 1.5 Euclidean distances and 0.2 distance relative error, the result of the exact range query will be null even if there is an object at a distance of just 1.5000001. Compared with the approximate static range query (ASR), $p11$ will be found as an interest object even at a distance of 1.6 and without increasing

**Algorithm 3.1.** Approximate static range query ASR algorithm

1: /* $q$: query point, $e$: the Euclidean distance threshold and $\gamma$ : the distance relative error */
2: /* Find approximate objects around the query $q$ */
3: Start from the root of the R-tree
4: The $upperbound = e \times (1+\gamma)$, $lowerbound = e \times (1-\gamma)$
5: LIST *resultlist*
6: **if** you access an internal node, **then**
7: calculate $MinDist(Ri,q)$ between $q$ and each possible MBR *Ri*. Insert into the Q-queue and sort them in ascending order of $MinDist$. Following this order, search downwards recursively only for those MBRs having $MinDist(Ri,q) \leq upperbound$
8: **end if**
9: **if** you access a leaf node **then**
10: calculate $MinDist(pi,q)$ between $q$ and each possible point ($pi$) stored in the node.
11: **if** $MinDist(pi,q) \leq upperbound$ **then**
12: $resultlist.add(pi)$, remove the root of the Q-queue, updating this data structure
13: **end if**
14: **end if**
15: **if** the Q-queue is empty **then**
16: stop
17: **end if**
18: Get the item $i$ on top of the Q-queue $< MinDist(Ri), Addrp >$
19: **if** this item has $MinDist(Ri,q) \leq lowerbound$ **then**
20: repeat the algorithm from $step$ 6 for this item
21: **end if**
22: **if** $MinDist(Ri,q) \leq upperbound$ and $resultlist.isempty()$ **then**
23: repeat the algorithm from $step$ 6 for this item
24: **else**
25: stop
26: **end if**

the search time. This gives the user an alternative option without having to conduct another search.

## 3.3 Approximate Moving Range Query based on Euclidean Distance

The main problem of the moving range query based on Euclidean distance is the multiple number of communications between the mobile device and the database server. Moreover, the users cannot make a decision due to the fast updating of the search result in areas with a high density of objects. To solve such problems, we considered the approximate concept and proposed two approximate moving range query AMR techniques. With the first technique, we reduced the range search to eliminate the number of the critical objects and also to minimise the search time. With the second technique, we reduced the number of split points, and this contributes in reducing both the number of times that results are updated and the number of communications with the server, thereby giving the user sufficient time to make a decision.

Full details of our proposed AMR approach and its algorithm are presented below (Section 3.3.2 and 3.3.3) starting with a brief background of the moving range query.

### 3.3.1 Moving Range Query

The idea of the moving range query is to divide the path into sub-segments and then implement the range search at both ends of each segment, where each retrieved object will have some corresponding split nodes which define the valid interval for the objects of interest (Xuan, Zhao, Taniar, Rahayu, et al., 2011). The following subsections explain in detail the steps involved in finding objects of interest using the moving range query approach.

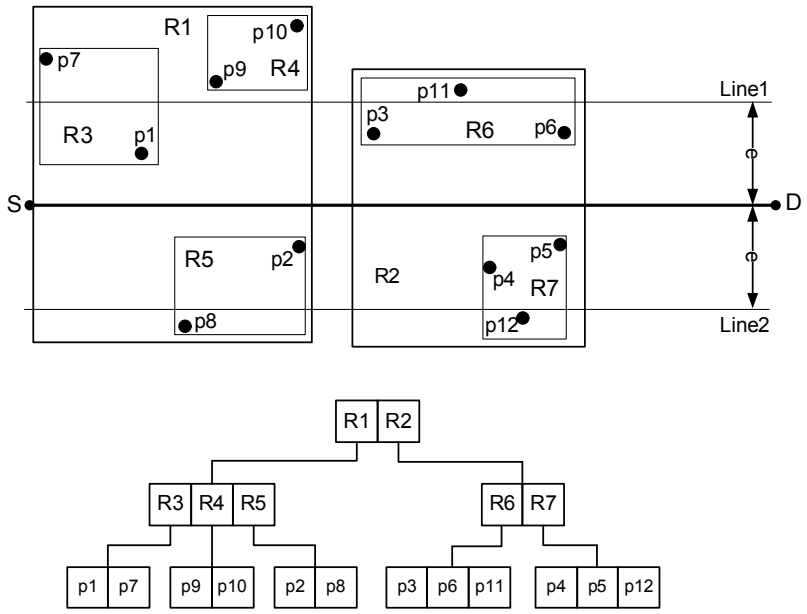



Figure 3.2: Filter step and refinement step in moving range query

**Filter Step and Refinement Step**

Generally, moving range query requires three steps to find the objects of interest: *filter step, refinement step* and *split step*. The filter step and refinement step in the moving range query is similar to the filter step and refinement step in static range query (see Section 2.1.1 Static Range Query). However, in the static range query, the query $q$ is a point and when the search starts traversing the R-tree, then a prune will occur to eliminate all MBRs with *MinDist* to the query point $q$ that are greater than $e$. Whereas, in the moving range query method the query $q$ is a path ($[S,D]$) where all MBRs with *MinDist* to $q = [S,D]$ greater than $e$ will be pruned.

Figure 3.2 shows an example of moving range query in Euclidean distance which clearly explains the filter step and refinement step. $SD$ is a segment of a road which represents the query path $q = [S,D]$; starting at $S$ and ending at $D$. Two lines within $e$ distance from the query path are generated to be parallel

with $q$. With the filter step and refinement step, all objects that fall above *line1* or under *line2* are pruned and the result is ($p1, p2, p3, p4, p5$ and $p6$).

### Split Step

The third step in the moving range query is the split step, which is used to determine at which point the user will obtain a new result. This result either has a new object of interest which enters the range search, or an object becomes out-of-date (i.e., leaves the range search) and should be removed from the result.

All objects of interest that pass the refinement step should get through the split step. In this step, two split points ($si$, $di$) are determined on the query path for each interest object $pi$ within the range search. The two split points ($si$, $di$) create a segment on the query path, where ($si$, $di$) represent the start and the end of the segment respectively. At each splitting point, the result should be updated when an object enters or leaves the range. For each split point $si, di \in SP$ $(1 \leq i \leq RL)$: $si, di \in q$, all points in segment ($si$, $di$) will consider $pi$ as one of the interest objects, where $SP$ is the split points list and $RL$ is the result list which contains all interest objects within the range search from the query path that passed the refinement step.

Figure 3.3 (i.e., derived from Figure 3.2) depicts how the split step works. The split step determines two split points for each object of interest in $RL$ by drawing a circle with a radius $e$ around each object $pi$, where $pi$ is its centre. Because the distance of the object $pi$ to the query path is not greater than $e$, the circle will then intersect the query path at two points ($si$ ,$di$). $\forall si, di \in SP\{s1, s2, ..., sr, d1, d2, ..., dr\}$, $dist(S, si) \leq dist(S, di)$: $1 \leq i \leq r$ , where $r$ represents the number of objects of interest in $RL$. $si$ and $di$ represent the location where the object enters and leaves the range search respectively (i.e., ($s1$, $d1$) are the split points of $p1$, ($s2$, $d2$) are the split points of $p2$ and

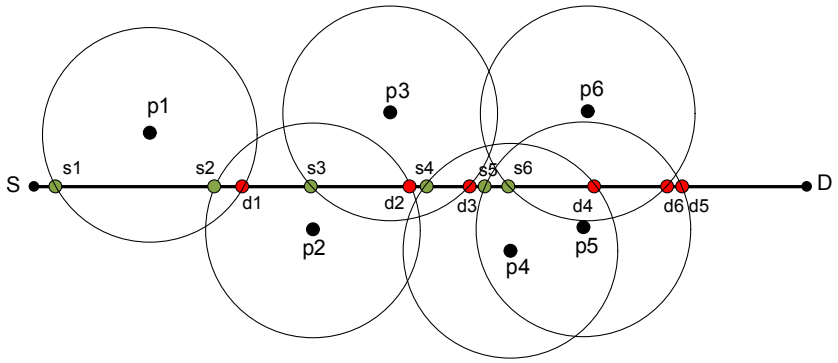



Figure 3.3: Split step in moving range query

so on). To avoid scanning the databases repeatedly, all split points with their corresponding coverings are reported (Tao et al., 2002). The split point list $SP$ has the start point $S$ if there is at least one object within its range.

According to Figure 3.3, the query path $[S,D]$ is divided into a number of segments. In this figure, there are two types of segments created on the query path, namely object segment and result segment. The object segment is created by an object of interest in the location where it enters and leaves the range search. For each object segment $si,di \in q,pi \in RL$. For example, within the object segment $(s1,d1)$, $p1$ will remain in the result list from $s1$ to $d1$, even when the moving query passes any number of split points. Hence, $p1$ will be safe within this segment. On the other hand, the result segment which is created by either intersection between two object segments (i.e., $(s1,s2)$ or $(s2,d1)$), or the gap between an object segment and its neighbour (i.e., $d1,s3$), will remain unchanged while the moving query is within the result segment, because no object is entering or leaving the range. Consequently, at each split point the result can only be changed.

For example, in Figure 3.4, the segments $((s1,s2)$, $(d2,d1)$ and $(d1,s3))$ represent the result segments, while the segments $((s1,d1)$, $(s2,d2)$ and $(s3,d3))$ represent the object segments. $p1$ is the result in the result segment $(s1,s2)$ and

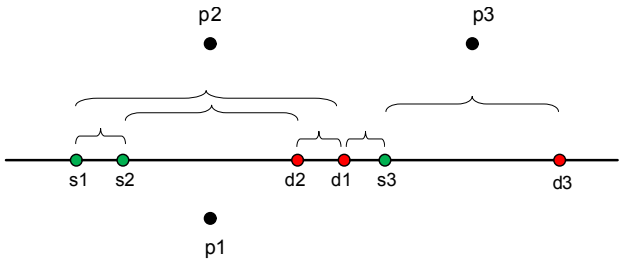



Figure 3.4: Object segment and result segment

will remain the same until the user enters the object segment $(s2, d2)$, and then the result will be $(p1, p2)$. With the continuous movement of the user, reaching the result segment $(d2, d1)$ will make the result become $(p1)$. It is very obvious that the object $p1$ remains in the result list while the query is moving within the object segment $(s1, d1)$, and $p2$ remains in the result while the query is in the object segment $(s2, d2)$ and so on. Therefore, after finding out all split points, the communication between the mobile device and the database server will occur only when the query reaches a split point to update the result list.

Figure 3.5 shows how the moving user invokes a query on each split point in order to be up-to-date. To distinguish between the object that comes within the range and the one that goes beyond the range, we draw the query range in green and red colours to indicate entering and leaving the range respectively.

**Example**

To better understand the function of the moving range query, simply consider Figure 3.5 as an example of the moving range query on R-tree, where $P$ is a dataset of interest objects $(P = \{p1, p2, ..., p6\})$. The output of the range query, for the specific path (line segment $[S, D]$) and the range query search $e$, is $\{(p1, s1, d1), (p2, s2, d2), (p3, s3, d3), (p4, s4, d4), (p5, s5, d5), (p6, s6, d6)\}$. That means the interest object $p1$ is one of the interest objects for interval $(s1, d1)$, and $p2$ is one of the interest objects for interval $(s2, d2)$ and so on.

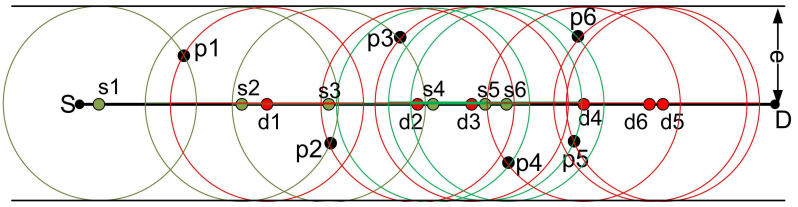



Figure 3.5: Moving range query

In the moving range query the points on the query segment (i.e. $s2, s4, d1$) represent a change in the result of the interest objects within the range query $e$ and these points are called split points.

### 3.3.2 Approximate Moving Range Query (AMR) Techniques

In our approximate moving range query AMR approach, we address the shortcomings of moving range query based on Euclidean distance by giving the users sufficient time to make a decision that fulfils their needs and is therefore appropriate. This is achieved by reducing the number of communications with the server, i.e., reducing the number of times that results are updated. In the AMR, we also reduce the number of split points by grouping and merging them with their neighbours to obtain new split points. Our AMR reduces the number of split points up to ten times less than moving range query.

Using the approximation concept, all objects whose distances are within an approximate range $e$ along the query path $q = [S, D]$ are retrieved. This requires some search time which is, in turn, affected by the number of object(s) of interest that have been checked, the size of the range search $e$, and the trajectory's length of the query $q$. On the other hand, to avoid scanning the databases many times, all split points with their corresponding coverage are reported (Tao et al., 2002). Note that the split point list $SP$ has the start point $S$ if there is at least one object within its range.

Two models of AMR queries are proposed, *Range Search Minimisation* and *Split Points Minimisation* to reduce the communications with the server and to reduce the number of split points. The following subsections explain these two methods in detail.

**Range Search Minimisation**

With this method, the boundary of the range query (range search) will be variable but not fixed as the exact range search. The variable range gives the search more flexibility in finding the object(s) of interest on R-tree. Therefore, by reducing the area of the range search, objects of interest can be found within a short time.

Here, we have used the *Lowerbound* ASR method (i.e., illustrated in Section 3.2.1) to reduce the search time by retrieving fewer results. This is achieved by avoiding object of interest retrievals that fall away from the query $q$ (especially in a high density environment), consequently giving users the result in a short time.

To find the objects of interest in this search, a filter step and refinement step are required. Figure 3.6 shows the function of these two steps, where $[S,D]$ is the query path. The dark shaded area represents the *lowerbound* distance to the query path and the light shaded area is the exact range search $e$. The internal nodes ($R3, R4$ and $R6$) will be expanded looking for objects of interest, while the internal node $R5$ will be ignored because its minimum distance to the query path is greater than the *lowerbound*. When the intermediate nodes ($R3, R4$ and $R6$) are expanded, any object within the range search $e$ will be retrieved. The result of this expansion will be represented by the objects ($p2, p3, p5, p6$ and $p10$). This result will then enter the third step in this search, the *split step*, which is used to determine at which point the user will obtain a new result.

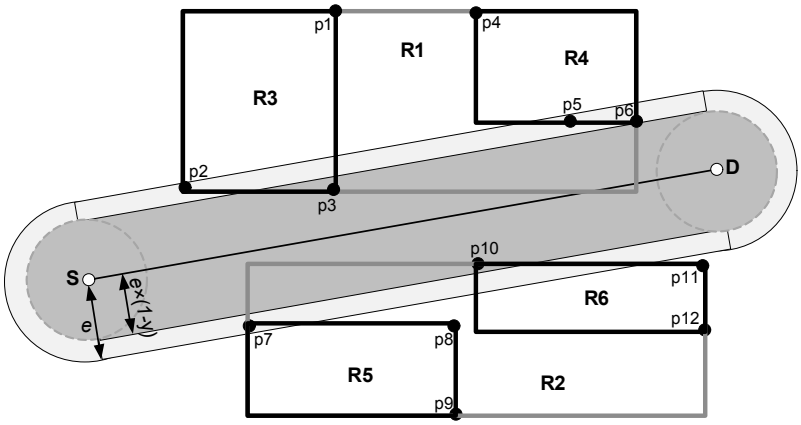



Figure 3.6: Approximate moving range query on R-tree

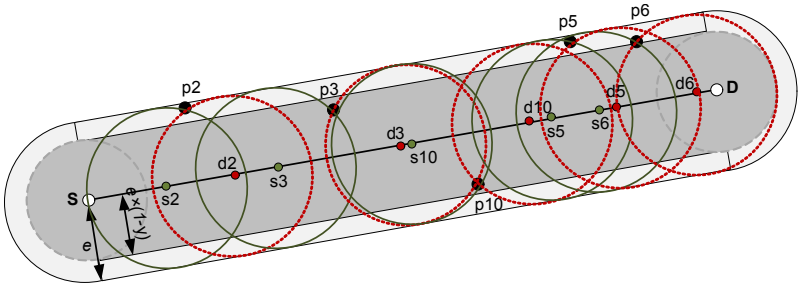



Figure 3.7: Range Search Minimisation in approximate range query

Figure 3.7 depicts the *split step*, where the result of the query will be produced as the following $\{(p2, s2, d2)$, $(p3, s3, d3)$, $(p10, s10, d10)$, $(p5, s5, d5)$, $(p6, s6, d6)\}$. Consequently, interest object $p2$ is one of the interest objects for interval $(s2, d2)$, and $p3$ is one of the interest objects for interval $(s3, d3)$ and so on.

In AMR, the points on query segment (i.e. $s2, d2, s3$) represent the split points. At each split point, the result will be updated and a communication between the mobile user and the database server will occur. For example, at $s3$ object $p3$ will be in the result list while at $d3$ the result will be updated and the object $p3$ will be removed from the result.

Approximate moving range queries have the same characteristics as moving range queries, in addition to other characteristics that make AMR faster in

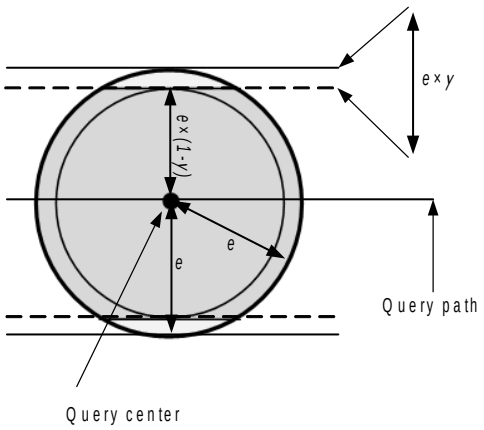



Figure 3.8: The range area of approximate moving range query

terms of the process, whereby AMR also provides alternative solutions to let the user make a decision without conducting another search. Also, the number of communications with the server is very low when performing AMR.

This method is mainly intended to reduce the area of the search range which will lead to exclusion of some objects from the result. That means that those objects which fall far from the query path will be excluded. Consider Figure 3.8 as an example, where the dark shaded area represents the range search of the moving query and gives the exact result, and the light area, $e \times \gamma$, represents the approximate result. Any object in the approximate area will be retrieved as an object of interest, only if that object is located in the internal node (MBR) and part of this internal node is within the dark shaded area.

**Split Points Minimisation**

With this method, the number of the split points is reduced. This will reduce the number of calculations occurring at each split point and the number of communications with the server. Some split points will be merged and clustered with their neighbours in groups depending on their position. In each group, the distance between the first point (the nearest one to the start point "*S*") and the

Table 3.2: Euclidean distance between split points

| Euclidean distance | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S$ | $s1$ | $s2$ | $d1$ | $s3$ | $d2$ | $s4$ | $d3$ | $s5$ | $s6$ | $d4$ | $d6$ | $d5$ | $D$ |
| $Sp2Sp$ | 1 | 7.4 | 1.4 | 3.4 | 4.4 | 0.8 | 2 | 0.6 | 1 | 4 | 3.4 | 0.6 | 6 |
| $S$ | 1 | 8.4 | 9.8 | 13.2 | 17.6 | 18.4 | 20.4 | 21 | 22 | 26 | 29.4 | 30 | 36 |

last point in the group (the furthest one to the start point "$S$") should not be greater than the distance ratio error $\rho$, $\rho = (e \times \gamma)$. The mean (average) method will be used to cluster the split points.

Table 3.2 shows the Euclidean distance between each split point and the other split points that are in Figure 3.3. The second row ($Sp2Sp$) shows the distance between each split point and its neighbour. To give specific details, number 1 in the second row and second column represents the distance between $S$ and $s1$ and in the third column, 7.4 represents the distance between $s1$ and its neighbour $s2$ and so on. The third row represents the distance between each split point and the start point ($S$) (i.e., number 13.2 represents the distance from the $S$ to the split point $s3$).

According to Figure 3.3 and Table 3.2, if the range search is $e = 10$ and the distance relative error is $\gamma = 0.2$, then $\rho = (10 \times 0.2) = 2$. $s2$ and $d1$ will be clustered together because the distance between $s2$ and $d1$ is less than $\rho$ ($1.4 < 2$), and a new split point will be calculated. Figure 3.9 shows the location of the new split point after the *minimise split point* method has been applied as in Figure 3.3.

**Heuristic1**. Given more than one split point (e.g. $spi$, $spj$, $spx$, ...) fall within $\rho$ distance, a mean (average) distance among them is calculated by the following equation:

$C(spi, spj, spx, ...) = dis(S, spi) + (dis(spi, spj) + dis(spj, spx) + ...)/$ *number of split point*

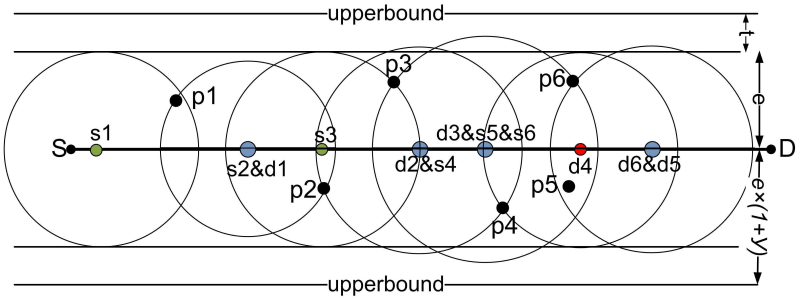



Figure 3.9: Split Points Minimisation in approximate moving range query

When $s2$ and $d1$ are clustered, $C(s2,s1)$, the result will be a new split point: $C(s2,d1) = dis(S,s2) + (dis(s2,d1))/2 = 8.4 + (1.4)/2 = 9.1$

The new split point $s2\&d1$ will be located at 9.1 from the start point $S$. That means at this new split point the object $p2$ will be within the range search, while the object $p1$ will leave the range search.

**Heuristic2**. Clustering two split points $spi$, $spj$ together will shift $spi$ by $\rho/2$ and $spj$ by $-\rho/2$ as a maximum, where $spi$ is the nearest to the start point $S$ from $spj$, $dis(S,spi) \leq dis(S,spj)$.

Assume that $dis(S,spi) = R$ and $dis(S,spj) = Q$, with regard to Heuristic 1, $Q - R \leq \rho$, the distance among the objects to be clustered should not be greater than $\rho$. $C(spi,spj) = dis(S,spi) + (dis(spi,spj))/2$ if the distance between $R$ and $Q$ is the maximum ($\rho$), then $C(spi,spj) = R + (Q-R)/2 \implies C(spi,spj) = R + (\rho)/2$. The $spi$ will be shifted by $\rho/2$, while $spj$ will be shifted by $-\rho/2$.

The main concept behind this method is that all objects within distance $e$ from the query path will be retrieved as a result. However, this result will be updated approximately within a split point, and this will make some objects of interest enter the range search earlier (i.e., $MinDist(pi,q) > e$) or later (i.e., $MinDist(pi,q) < e$ ) while other objects may leave the range search earlier or later (i.e, before or after becoming out of date).

**Algorithm 3.2.** Range Search Minimisation algorithm based on Euclidean distance

1: /* *SD*: the segment query path ($q$), $e$: the range search and $\gamma$: the distance relative error */
2: /* Find approximate objects around the query $q$ and their split points */
3: Queue Result $QR = \phi$
4: LIST *SP*
5: $lowerbound = e \times (1 - \gamma)$
6: start from the root of the R-tree
7: find all candidates MBR $R$, $MinDist(R,q) \leq lowerbound$
8: **for** each candidate MBR **do**
9: find an interest object $pi$, where $MinDist(pi,q) \leq e$
10: **for** each interest object $pi$ **do**
11: find two split points, $s_i$ and $d_i$, on the segment *SD*, where $D_E(s_i,pi) = D_E(d_i,pi) = e$
12: $SP.add(s_i)$, $SP.add(d_i)$
13: $QR = QR \cup (pi$, start split node $s_i$, end split node $d_i)$
14: **end for**
15: **end for**

### 3.3.3 AMR Algorithms

Two algorithms are described below, *Range Search Minimisation* and *Split Points Minimisation*, based on two approaches introduced in the previous section. These two algorithms use branch-and-bound techniques to prune unqualified internal nodes (MBR) on the R-tree in order to reduce the search space.

#### Range Search Minimisation Algorithm

In this algorithm (Algorithm 3.2) we minimise the range search using the R-tree to index the objects with high performance. A query segment (path) and a range search $e$ have been given to find all objects of interest, such as post offices, within this range along the query segment. The distance relative error $\gamma$ is also given and represents the tolerance of the distance error that is acceptable for the user. In this method the range search will be adjusted to be equal to $lowerbound = e \times (1 - \gamma)$ to reduce the search area.

The algorithm of Range Search Minimisation works as follows: each intermediate node, $R$, should be determined as a candidate and then be expanded if $MinDist(R,q) \leq e \times (1-\gamma)$. That means this internal node is within the range search of $e$, in a specific segment of the query path. The algorithm determines whether the qualified object from a candidate internal node that has a minimum distance to query path $q$ is less than or equal to $e$, and then places this node in the queue result list $QR$.

For each qualified object, two split points will be determined: first $si$ when this object becomes within the range search $e$ from $q$, and second the split point $di$ when this object leaves the range search. For each split point, in or out, the result list should be updated, because there is at least one object that either leaves or enters the range search.

**Split Points Minimisation Algorithm**

With the Split Points Minimisation algorithm (Algorithm 3.3) all of the internal nodes within minimum distance from the query path $q$ less than or equal to $e$ will be considered as candidates. All these candidates will be expanded looking for objects of interest that have a maximum Euclidean distance $e$ to the query path. In this method the range search is fixed and equal to $e$.

After finding all split points for objects of interest, two or more split points can be merged. If these points fall within distance $\rho$, $\rho = e \times \gamma$ then the mean distance among them will be taken, to create a new split point. The mean distance among them will be calculated.

**Algorithm 3.3.** Split Points Minimisation algorithm based on Euclidean distance

1: /* $SD$: the segment query path ($q$), $e$: the range search and $\gamma$: the distance relative error */
2: /* Find all objects around the query $q$ and their approximate split points */
3: Queue Result $QR = \phi$
4: LIST $SP$
5: $\rho = e \times \gamma$
6: start from the root of the R-tree
7: find all candidates MBR $R$, $MinDist(R,q) \leq e$
8: **for** each candidate MBR **do**
9: find an interest object $pi$, where $MinDist(pi,q) \leq e$
10: **for** each interest object $pi$ **do**
11: find two split points, $s_i$ and $d_i$, on the segment $SD$, where $D_E(s_i,q) = D_E(d_i,q) = e$
12: $SP.add(s_i)$
13: $SP.add(d_i)$
14: $QR = QR \cup (pi$, start split node $s_i$, end split node $d_i)$
15: **end for**
16: **end for**
17: **for** each split point in $SP$ **do**
18: group split points within distance $\rho$
19: **for** each $Group_i$ of split points **do**
20: new split point $nsp_i$ = mean distance of $Group_i$
21: update $QR$, replace $nsp_i$ with the $Group_i$
22: **end for**
23: **end for**

## 3.4 Approximate Static Range Query based on Road Network Distance

In spatial network databases, research (Papadias et al., 2003; Safar, 2005; Taniar et al., 2011; Tran et al., 2009; Zhao et al., 2013) has focussed on developing efficient algorithms that expand the spatial query processing methods by integrating connectivity and location information. On this basis, two methods were developed, Euclidean Restriction and Network Expansion, for processing the most common spatial queries in spatial network databases, i.e., range query,

nearest neighbour, closest pairs, point location and distance joins (Bern, 1993; Papadias et al., 2003).

Compared to Euclidean distance ($D_E$), network distance ($D_N$) computations are significantly more expensive because they entail shortest path algorithms in large graphs (Papadopoulos et al., 2011; Safar, 2008). Consequently, implementing range queries in spatial network databases, namely Range Euclidean Restriction RER and Range Network Expansion RNE, requires lengthy calculation times to obtain the essential results. Also, when the search is finalised with a result of a massive number of objects or with absolutely no results, then users need to submit another query which means that another search is performed. Furthermore, repeated false hits (i.e., irrelevant objects as candidates) that occur during the search should be considered since each false hit represents a waste of search time. Therefore, an essential pre-request for solving these problems is to refine the processing of spatial range queries in spatial network databases.

We have designed two new methods, called *Approximate Range Euclidean Restriction (ARER)* and *Approximate Range Network Expansion (ARNE)* to eliminate the problems of the former methods, i.e., Range Euclidean Restriction RER and Range Network Expansion RNE. In these two techniques, we use *Lowerbound* ASR (presented in Section 3.2.1) which minimises the actual range search to exclude the internal nodes that fall outside the *lowerbound*. We also improve the selectivity of the filter step to reduce the number of the candidate objects, and consequently, minimise the number of communications between the mobile device and the database server. The resulting techniques achieve a better running time and deliver a better performance, yet with low false hits and reasonable false misses. To the best of our knowledge, this is the first work dealing with the efficient processing of approximate spatial range search queries in spatial network databases.

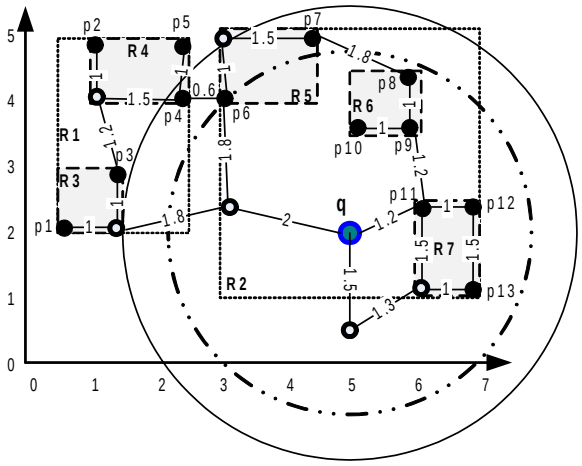



Figure 3.10: ARER in the road network

### 3.4.1 Approximate Range Euclidean Restriction (ARER)

The main characteristic of the ARER is the variability of its range which gives the search more flexibility to find the interest object(s) on the R-tree. In this method there are two boundaries to surround the search, the first one is the *exactbound* range search $e$ and the second one is the *lowerbound* $(e \times (1-\gamma))$. All objects that are retrieved as candidates should belong to MBRs which overlap with the *lowerbound*.

The ARER method returns a set of candidate objects $C$ within (Euclidean) *lowerbound* distance from $q$. $C$ probably misses a small number of objects, which are called false misses (i.e., $D_E(pi,q) > e \times (1-\gamma) \Longleftrightarrow D_N(pi,q) < e$ or $D_N(pi,q) \geq e$), and contains a small number of false hits.

Figure 3.10 represents an example of a spatial road network; Table 3.3 is the road network distance of Figure 3.10. Consider them as an example. Where $P$ is a dataset of interest objects $(P = \{p1, p2, ..., p13\})$, $q$ is a query point and $e = 3.5$ is a range search. If the RER is applied the candidate objects are: $C = \{p1, p2, ..., p13\}$ (filter step) depending on Table 3.1, and the result of the interest objects within network distance 3.5 from the query point is

Table 3.3: The network distance between objects and the query point $q$ in Figure 3.10

| Network distance from $q$ | | | |
|---|---|---|---|
| **object** | **distance** | **object** | **distance** |
| $p1$ | 4.8 | $p8$ | 3.4 |
| $p2$ | 6.9 | $p9$ | 2.4 |
| $p3$ | 4.8 | $p10$ | 3.4 |
| $p4$ | 4.4 | $p11$ | 1.2 |
| $p5$ | 5.4 | $p12$ | 2.2 |
| $p6$ | 3.8 | $p13$ | 3.7 |
| $p7$ | 5.2 | | |

($< p11, 1.2 >$, $< p12, 2.2 >$, $< p9, 2.4 >$, $< p8, 3.4 >$, $< p10, 3.4 >$) depending on Table 3.3 (refinement step).

When ARER is applied the candidate objects are: $C = \{p6, p7, ..., p13\}$, which represents the filter step. These candidates will go though the refinement step and the result will be ($< p11, 1.2 >$, $< p12, 2.2 >$, $< p9, 2.4 >$, $< p8, 3.4 >$, $< p10, 3.4 >$).

### 3.4.2 Approximate Range Network Expansion (ARNE)

The Approximate Range Network Expansion (ARNE) technique has two steps: qualifying segments and retrieving entities. The first step, qualifying segments *QS*, expands all nodes within *lowerbound* $(e \times (1 - \gamma))$ from query point $q$ searching for qualified segments. The qualified segments represent all or part of the segments that are located within the network distance *lowerbound* from the query point $q$. The second step, retrieving entities, evaluates the data entities falling within these segments in *QS* to obtain the actual result. Only R-tree nodes that overlap with qualifying segments will be visited. All MBRs that intersect with *QS* will be retrieved when searching for objects within the range search $e$ from the query point $q$.

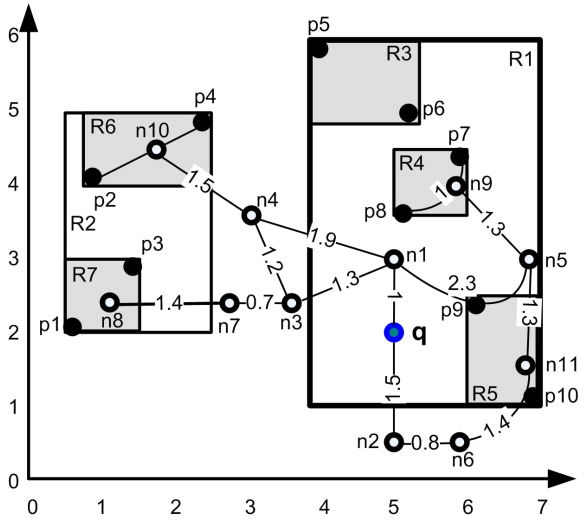



Figure 3.11: ARNE in the road network

Table 3.4: The network distance between objects, nodes and the query point $q$ in Figure 3.11

| Network distance from $q$ to | | | | | | | |
|---|---|---|---|---|---|---|---|
| **node** | **distance** | **node** | **distance** | **object** | **distance** | **object** | **distance** |
| $n1$ | 1 | $n7$ | 3 | $p1$ | 5 | $p5$ | - |
| $n2$ | 1.5 | $n8$ | 4.4 | $p2$ | 5.4 | $p7$ | 4.9 |
| $n3$ | 2.3 | $n9$ | 4.6 | $p3$ | 5 | $p8$ | 5.3 |
| $n4$ | 2.9 | $n10$ | 4.4 | $p4$ | 5.2 | $p9$ | 2.3 |
| $n5$ | 3.3 | $n11$ | 3.7 | $p5$ | - | $p10$ | 3.5 |
| $n6$ | 2.3 | | | | | | |

Consider Figure 3.11 and Table 3.4 as an example of ARNE. Where $P$ is a dataset of interest objects ($P = \{p1, p2, ..., p10\}$), $N$ is a set of intersection nodes ($N = \{n1, n2, ..., n11\}$), $q$ is a query point, $e = 3.5$ is a range search and $\gamma = 0.2$ is a distance relative error.

The qualifying segments of ARNE are ($< n1, n5 >$, $< n5, n11 >$, $< n6, n11 >$, $< n5, n9 >$) which overlap with MBRs, $R4$ and $R5$, and their network distance from $q$ is less than the *lowerbound*. All objects within the MBRs ($R4$ and $R5$) will be examined to find the objects that are within network distance $e$ from the $q$. The result is $p9$ and $p10$.

**Algorithm 3.4.** Approximate Range Euclidean Restriction ARER algorithm

```
1: /* q: query point, e: the Euclidean distance threshold and γ : the distance
   relative error */
2: /* Find approximate objects around the query q */
3: LIST resultlist, cand
4: cand = call Lowerbound ASR algorithm (q, e, γ)
5: segment s(n_i,n_j) = find-segment(q)
6: Q =< (n_i,D_N(q,n_i)),(n_j,D_N(q,n_j)) >
7: de-queue the node n in Q with the smallest D_N(q,n)
8: while (D_N(q,n) ≤ e and cand.isempty()) do
9:    for non-visited adjacent node n_x of n do
10:      for all points pi in cand do
11:         if pi exists in segment (n_x,n) and D_N(q,n)+D_N(n,pi) ≤ e then
12:            resultlist.add(pi)
13:            cand.remove(pi)
14:            en-queue (n_x,D_N(q,n_x)) in Q
15:            de-queue the next node n in Q
16:         end if
17:      end for
18:   end for
19: end while
```

On the other hand when RNE is applied the qualifying segments are ($< n1,n5 >$, $< n5,n11 >$, $< n6,n11 >$, $< n5,n9 >$, $< n7,n8 >$, $< n4,n10 >$). The MBRs that will be visited are ($R4, R5, R6$ and $R7$) and the result is the same as ARNE.

### 3.4.3 ASR Algorithms in the Road Network

Full details of the Approximate Range Euclidean Restriction (ARER) algorithm and Approximate Range Network Expansion (ARNE) algorithm are presented below.

#### ARER Algorithm

In the ARER algorithm (Algorithm 3.4), the list (*cand*) represents the candidate objects when executing the *Lowerbound* ASR algorithm (i.e., filter step). To

**Algorithm 3.5.** Approximate Range Network Expansion ARNE algorithm

```
1: /* q: query point, e: the Euclidean distance threshold and γ : the distance
   relative error */
2: /* Find approximate objects around the query q */
3: while D_N(q,n) < e × (1 − γ) do
4:    Expand node n to all adjacent nodes (segments)
5:    for each adjacent node n_x do
6:       if n_x has not been visited then
7:          D_N(q,n_x) = D_N(q,n) + D_N(n,n_x)
8:       end if
9:       if D_N(q,n_x) ≤ e then
10:         QS = QS ∪ {(n, D_N(q,n)), (n_x, D_N(q,n_x))}
11:      else
12:         QS = QS ∪ {(n, D_N(q,n)), (n'_x, D_N(q,n'_x))}
13:         at the position where D_N(q,n'_x) = e
14:      end if
15:   end for
16: end while
```

determine the actual result, the candidate objects that pass the filter step will be compared with the road network representation of the segment (i.e., refinement step). Each object which passes the filter step (candidate) will be tested against its segment. Then, each segment containing a candidate object will be visited when at least one of its ends will be within the distance of $e$ from the query.

### ARNE Algorithm

In the ARNE algorithm (Algorithm ARNE Algorithm), the search starts from the root of the R-tree to visit all the nodes that intersect with the MBRs. In this algorithm, the approximate static range query (ASR) method is used to find the candidate objects (filter step), which means, only the segment with at least one end point within distance $(e-\gamma)$ from the $q$ will be expanded. Expanding this segment will continue, but no more expanding will occur if the second end of the segment is larger than $(e+\gamma)$.

## 3.5 Approximate Moving Range Query based on Road Network

The main problems of the moving range query, which is based on the spatial road network, are: the high amount of network expansion, the continuous query re-revaluation and the multiple number of communications between the user, who invokes the query, and the database server. In this section, we will introduce our proposed method which eliminates what the moving range query is suffering from in the road network, but first we will start with some background information.

### 3.5.1 Moving Range Query - Preliminaries

The moving range query, which is based on a spatial road network, can be defined as: the retrieval of all objects of interest within a specific network distance around the user, who invokes the query while he is moving. This technique divides the complete query path into road segments. Each road segment connects two consecutive road intersections, and each single road segment should be processed individually. Thus, processing the segments one by one, looking for objects of interest starting from the query point, is the main idea of accumulating the whole segments within the radius of the range search.

In the road network authors in (Xuan, Zhao, Taniar, Safar, & Srinivasan, 2011) introduce a continuous range query, which is an architecture that integrates split points and Range Euclidean Restriction *RER*. Their technique is based on creating a search region for the moving query which expands from each intersection and is similar to Dijkstra's algorithm (Dijkstra, 1959). The advantage of this technique is to find the exact network distance in which the

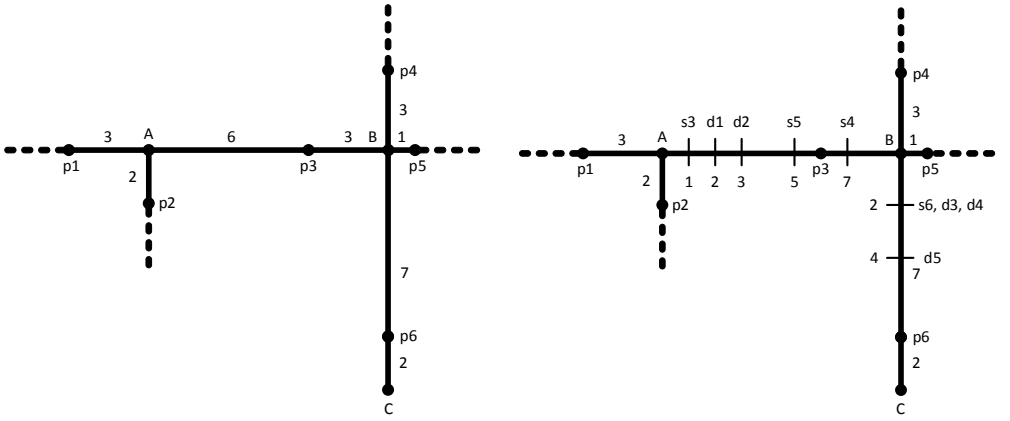
moving query result will be updated. However, this technique indicates that for each object of interest within a road segment there is only one contributing split point, and there the object will either exit or enter the range. But, in the case of an object that is located between two consecutive intersections within a distance of more than the radius of the query from each side, then this indication will be false because the object will have two split points that connect the two intersections on the segment. In this section we will introduce the technique of the moving range query based on the road network with some modifications to override this weakness.

Consider Figure 3.12(a) as an example of the query. It shows a part of a big map or a longer query path. The query path goes first through the segment $AB$ and then goes through the segment $BC$ with both segments being part of a bigger path. Let's use these segments to illustrate an example of a moving range query. Assume there are six objects around the segments $AB$ and $BC$, ($p1$, $p2$, $p3$, $p4$, $p5$ , $p6$). The distance between the objects and the two sides of each segments are given (e.g. $D_N(A, p1) = 3$) beside the length of each segment.

The process of the moving range query based on a spatial road network is similar to the moving range query based on Euclidean space, where each object of interest needs to be rechecked continuously. Based on the distance between the object of interest and the query point, there are two categories of objects of interest that need to be processed:

1. objects of interest currently located within the boundary of the query, and getting farther apart from the moving query, and after some distance they will leave the boundary and become out of range.

2. objects of interest located outside the boundary, and getting closer to the moving query, and will soon get inside the boundary of the query.




(a) Moving range query

(b) Split points for moving range query where $e = 5$

Figure 3.12: Moving range query based on spatial road network

Let's assume that the radius $e$ of the moving range query is equal to 5, and the query is located on $A$ and moving toward $B$ first, and then moves toward $C$. For each of these objects, we need to calculate their split points. Let's start with the expired objects of interest that will be located behind the moving query, these objects should be excluded from the result list of the query. At some points the objects in the result list will expire because the query moves. To distinguish between the spilt points of the objects that join the query or leave it, we will refer to them by $si$ and $di$. $s$ means the split point of the object joins the query, $d$ means the split point of the object leaves the query, and $i$ is the object's number/name (see Figure 3.12(b)). By assuming the query is located on $A$, the objects $p1$ and $p2$ will expire at some points at $e - D_N(A, pi)$. So, $p1$ will expire at the distance of $5-3=2$ ($d1=2$), away from $A$ at the segment $AB$, when the radius of the range query is $e = 5$. As the query moves away from $A$, $p1$ is getting farther away. At distance 2 from $A$, $p1$ will be out of the query boundary. The same principle is applied to $p2$, in this case, $p2$ will expire at $5-2=3$ ($d2=3$). As one object is leaving the query results, a split

point is identified. In this case, two split points are identified and they are at position 2 and 3 from $A$.

Now, we should calculate the split points of the objects where the query is moving toward, for example, $p3$, $p4$ and $p5$. At some points along segment $AB$ these objects will be included in the result list of the query and these points represent the split points for these objects. *Note*, only objects located either on this segment or within distance $e$ from $B$ will be considered here, (e.g., $p6$ farther than 5 from $AB$ will be considered later). Because $p3$ is located on the segment $AB$ we need to calculate its split points from $A$, while for every other object, we need to calculate their split points from $B$. The object $p3$ will join the result list at $|e - D_N(A, p3)| = |5-6| = 1$ ($s3 = 1$) distance from $A$. The rest of the objects of interest, will be inside the boundary of the query precisely at $|e - D_N(B, pi)|$ from $B$. Objects $p4$ and $p5$ will be within the query at $|5-3| = 2$ ($s4 = 2$) and $|5-1| = 4$ ($s5 = 4$) from $B$ respectively at the segment $AB$. At the same time objects $p4$ and $p5$ will expire at the same distance from $B$ but they will be at the segment $BC$, $d4 = 4$ and $d5 = 4$ respectively.

However, the split points $(s4, s5)$ which are 2 and 4 distances away from $B$ need to be converted into a distance from $A$. In this case, the distance from $A$ is $D_N(A,B) - (|e - D_N(B, pi)|)$ at segment $AB$. Object $p4$ will be in the result list at the split point $9-2 = 7$ and $p5$ will be at $9-4 = 5$; both are from $A$.

Finally, when the query arrives at intersection $B$ the same procedure above will be applied on segment $BC$. Each object of interest will create two split points, one split point when the object is joining the result list of the query, and another split point when the object is leaving the result list of the query. Furthermore, more than one object could share the same split point (e.g., $p3$ and $p4$ are leaving the result list at distance 2 from $B$ and at the same time location $p6$ is joining the result).

Figure 3.12(b) shows that there are seven split points at distances 1, 2, 3, 5 and 7 from $A$ located at segment $AB$, and two split points at distances 2 and 4 from $B$ located at segment $BC$. The query results for segment $AB$ are:

$A = \{p1, p2\}$

$A \rightarrow s3 = \{p1, p2, p3\}$ $p3$ is entering at $s3$

$s3 \rightarrow d1 = \{p2, p3\}$ $p1$ is leaving at $d1$

$d1 \rightarrow d2 = \{p3\}$ $p2$ is leaving at $d2$

$d2 \rightarrow s5 = \{p3, p5\}$ $p5$ is entering at $s5$

$s5 \rightarrow s4 = \{p3, p5, p4\}$ $p4$ is entering at $s4$

$s4 \rightarrow B = \{p3, p5, p4\}$

While the query results for segment $BC$ are:

$B = \{p3, p5, p4\}$

$B \rightarrow s6 = \{p5, p6\}$ $p6$ is entering and $p3$, $p4$ are leaving at $s6$

$s6 \rightarrow d5 = \{p6\}$ $p5$ is leaving at $d5$

$d5 \rightarrow C = \{p6\}$

### 3.5.2 Approximate Moving Range (AMR) Techniques

In this section, we present a new technique to process the moving range query in road networks based on the concept of approximation. Our technique will reduce the range search to eliminate the number of critical objects and also to minimise the search time. In addition, this technique will minimise the number of split points in order to lower the communications rate and reduce the number of times that results are updated.

**Range Search Minimisation (based on Road Network)**

This method is similar to the Range Search Minimisation method which we proposed in Section 3.3.2. Some modifications are applied to make this method work with the road network:

1. Dividing the complete query path into road segments.
2. Expanding each segment to search for objects of interest using ARER which we proposed in Section 3.4.1.

**Split Point Minimisation (based on Road Network)**

Reducing the number of split points is done by merging and clustering some split points with their neighbours in groups. The distance among all the split points in any group will not be greater than the distance ratio error $\rho$, $\rho = (e \times \gamma)$, where $e$ is the range of the query and $\gamma$ is the error threshold. This method is similar to the split point minimisation (i.e., based on Euclidean distance) which we proposed in Section 3.3.2, however, the only difference is that we will use the Range Euclidean Restriction (RER) method to find the objects of interest. After finding the split points for the objects of interest, then we will group them together and the mean distance among them will be calculated. A single point (the mean's result) will replace a group of split points.

### 3.5.3 AMR Algorithms (based on Road Network)

Two algorithms for approximate moving range query based on the road network are described below, *Range Search Minimisation* and *Split Points Minimisation*, derived from two approaches introduced earlier in Section 3.3.2. These two

algorithms use branch-and-bound techniques to prune unqualified internal nodes (MBR) on the R-tree in order to reduce the search space.

**Range Search Minimisation Algorithm (based on Road Network)**

In this algorithm (Algorithm 3.6) we minimise the range search of the moving query. A query path and a range query search $e$ are given to find all objects of interest, such as petrol stations, within this range along the path of the moving query. The distance relative error $\gamma$ is given too, which represents the tolerance of the distance error acceptable from the user. R-tree is used to index the objects with high performance. In this method, the range search will be adjusted to be equal to $lowerbound = e \times (1-\gamma)$ to reduce the search area.

The algorithm of Range Search Minimisation works as follows: each intermediate node, $R$, should be determined as a candidate and then to be expanded if $MinDist(R,q) \leq e \times (1-\gamma)$. That means that this internal node may have object(s) of interest within the range search of $e$. The algorithm will include each candidate internal node looking for candidate object(s). Then the entire path will be divided into road segments; each road segment connects two consecutive road intersections. Each segment has a candidate object which will be expanded (i.e., see Section 3.5.1) while searching for objects of interest.

For each object of interest $pi$, two split points will be determined; first $si$ when this object becomes within the range search $D_N(e)$ from $q$, and second split point $di$ when this object leaves the range search after $D_N(e)$ from the query. For each split point, in or out, the result list should be updated, because there is at least one object that either leaves or enters the range search.

**Algorithm 3.6.** Range Search Minimisation algorithm based on road network

1: /* *SD*: the segment query path ($q$), $e$: the range search and $\gamma$: the distance relative error */
2: /* Find approximate objects around the query $q$ and their split points */
3: Queue Result $QR = \phi$
4: LIST *resultlist*, *SP*, *cand*
5: $lowerbound = e \times (1 - \gamma)$
6: start from the root of the R-tree
7: find all candidates MBR $R$, $MinDist(R,q) \leq lowerbound$
8: **for** each candidate MBR **do**
9: find an candidate object $pi$
10: **if** $D_E(pi,q) < e$ **then**
11: $cand.add(pi)$
12: **end if**
13: **end for**
14: divide the query path $[S,D]$ into segments
15: each segment has two ends $(r_j, r_{j+1})$
16: **for** each segment in the query path **do**
17: expand $r_j, r_{j+1}$ in all directions for $D_N(e)$
18: **for** each candidate object ($pi$) in *cand* **do**
19: **if** $D_N(pi, r_j(r_{j+1})) < e$ **then**
20: $resultlist.add(pi)$
21: **end if**
22: **end for**
23: **end for**
24: **for** each object of interest $pi$ in *resultlist* **do**
25: find two split points, $s_i$ and $d_i$, on the corresponding segment $r_j, r_{j+1}$, where $D_N(s_i, pi) = D_N(d_i, pi) = e$
26: $SP.add(s_i)$
27: $SP.add(d_i)$
28: $QR = QR \cup (pi$, start split node $s_i$, end split node $d_i)$
29: **end for**

### Split Points Minimisation Algorithm (based on Road Network)

With Split Points Minimisation algorithm (Algorithm 3.7) all the internal nodes within a minimum distance from the query path $q$ less than or equal to $e$, will be considered as candidates. All these candidates will be expanded in order to search for candidate objects with a maximum Euclidean distance $e$ to the query path. In this method, the range search is fixed and equal to $e$. After finding all

of the candidate objects, the entire query path will be segmented. Each segment with a candidate object(s) will be expanded using the RER method looking for objects of interest located within distance $D_N(e)$ from each intersection on the query path. Then for each object of interest, two split points will be determined. After finding all the split points, two or more split points can be merged. If these points fall within the distance $\rho$, $\rho = e \times \gamma$, then the mean distance among them will be calculated and considered as a new split point.

## 3.6 Experimental Results

Several experiments are carried out to evaluate our proposed algorithms. To address the limitation of Euclidean distance in the real world, a synthetic dataset is used to test our proposed *approximate static range query* ASR, and *approximate moving range query* AMR based on Euclidean distance, and *approximate static range query* ARER, ARNE, and *approximate moving range query* AMR based on network distance. We test 200 different queries using each approach.

In ASR we calculate the average number of false hits and interest objects that have been retrieved, then compare each of them with the exact result. On the other hand, in AMR we calculate the average number of split points and compare this with the exact result. We conduct two performance measures of ASR. First, the average number of interest objects returned as an answer are compared with the exact result. Second, a comparison is made between the average number of false hits in ASR and the exact range query (i.e., number of false hits indicates extra *I/O* access). Furthermore, we conduct one performance measure with AMR to minimise the number of split points. This is very important because it indicates the number of communications with the database server and the power consumption required by the user's device to update the

**Algorithm 3.7.** Split Points Minimisation algorithm based on road network

/* *SD*: the segment query path ($q$), $e$: the range search and $\gamma$: the distance relative error */
/* Find all objects around the query $q$ and their approximate split points */
Queue Result $QR = \phi$
LIST *resultlist*, *SP*, *cand*
$\rho = e \times \gamma$
start from the root of the R-tree
find all candidates MBR $R$, $MinDist(R,q) \leq e$
**for** each candidate MBR **do**
  find an candidate object $pi$
  **if** $D_E(pi,q) < e$ **then**
    $cand.add(pi)$
  **end if**
**end for**
divide the query path $[S,D]$ into segments
each segment has two ends $(r_j, r_{j+1})$
**for** each segment in the query path **do**
  expand $r_j, r_{j+1}$ in all directions for $D_N(e)$
  each candidate object ($pi$) in this segment,
  **if** $D_N(pi, r_j(r_{j+1})) < e$ **then**
    $resultlist.add(pi)$
  **end if**
**end for**
**for** each object of interest $p_i$ in *resultlist* **do**
  find two split points, $s_i$ and $d_i$, on the corresponding segment $r_j, r_{j+1}$, where $D_N(s_i, pi) = D_N(d_i, pi) = e$
  $SP.add(s_i)$
  $SP.add(d_i)$
  $QR = QR \cup (p_i$, start split node $s_i$, end split node $d_i)$
**end for**
**for** each split point in *SP* **do**
  group split points within distance $\rho$
  **for** each $Group_i$ of split points **do**
    new split point $nsp_i$ = mean distance of $Group_i$
    update $QR$, replace $nsp_i$ with the $Group_i$
  **end for**
**end for**

result. We select 500 random queries as facts and use them in high, medium, and low density environments in our experiments (i.e., 500 objects, 300 objects

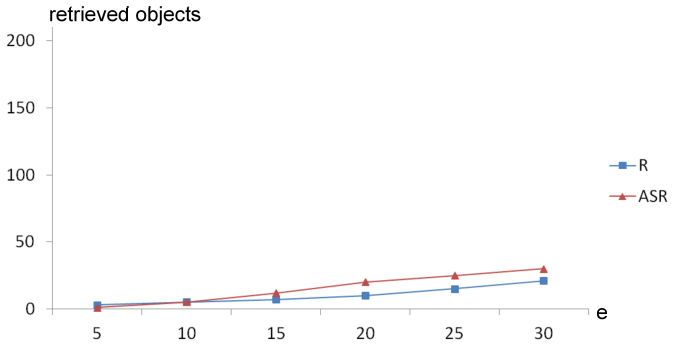
and 100 objects represent high, medium, and low density within $10000km^2$ respectively, assuming the distance relative error is $\gamma = 0.2$).

In *Lowerbound approximate static range query* (*Lowerbound* ASR) we calculate the number of candidate objects that have been retrieved on average together with the number of false hits on average, and compared each of them with the exact result. To examine our approach ARER, the result that we obtained from the *Lowerbound* ASR was applied on the Melbourne City road network provided by Whereis *(http://www.whereis.com/)*. The map of the Melbourne City is also used to examine ARNE. We measure the performance of *Lowerbound* ASR in respect to: *i*) the average number of candidate objects in the answer, and their similarity to the exact result, and *ii*) a comparison between the average number of false hits (i.e., extra I/O access ) in *Lowerbound* ASR and the exact range query. We select 200 random queries in various distributions and range sizes. To create the average result, 100 objects, 300 objects and 500 objects represent the low, medium and high density environments respectively. In addition, we use a range search from $0.5km$ to $3km$ within $10000km^2$. We also assume the distance relative error $\gamma = 0.2$.

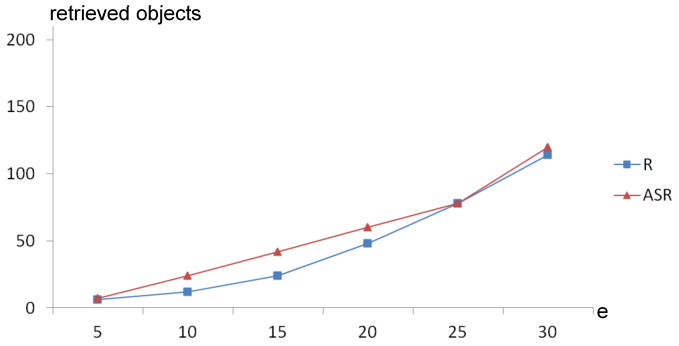
### 3.6.1 Query based on Euclidean Distance

#### Retrieved Objects and False Hits in ASR

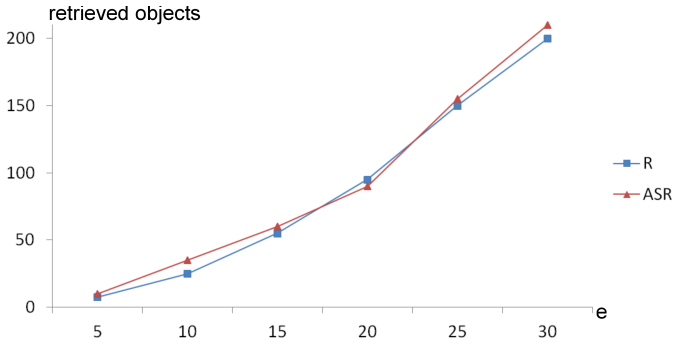
Figure 3.13 depicts the performance of our algorithm ASR with a range varying search from $0.5km$ to $3km$. We find that the effect of the increasing range search on the number of retrieved objects is similar between the range query and ASR. Also, the retrieved objects in ASR are slightly more than the retrieved objects in the range query, but false hits in the range query are two times the false hits in ASR on average. Furthermore, the accuracy of the result using ASR




(a) Low density 100 objects



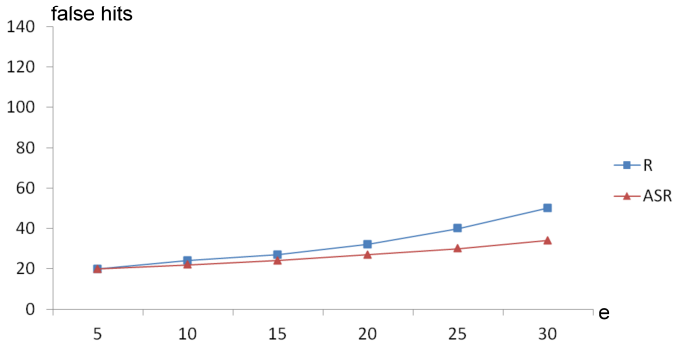

(b) Medium density 300 objects



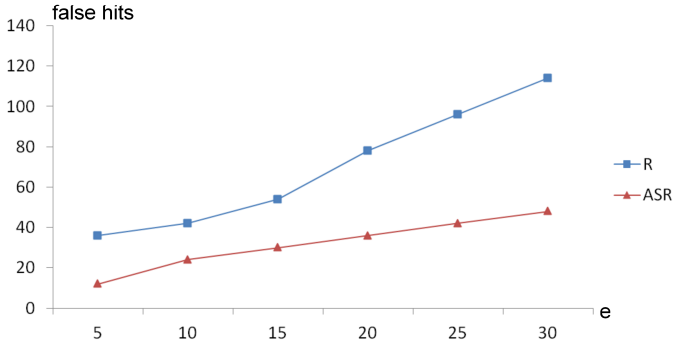

(c) High density 500 objects

Figure 3.13: Retrieving interest objects in ASR compared with range query (R)

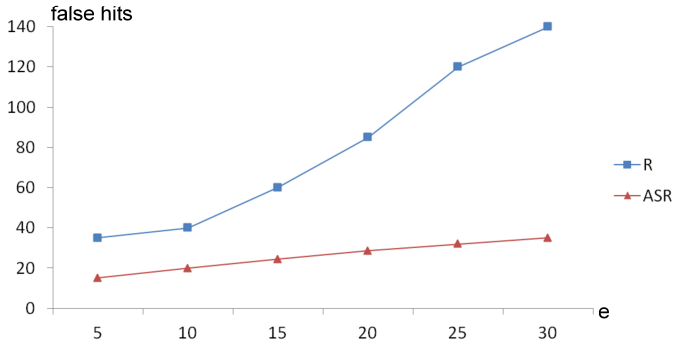
compared with the exact result is equal to 85%, 88% and 94% in low, medium, and high density respectively. Figure 3.14 depicts the percentage of false hits is 52%, 26% and 23% in low, medium and high density respectively, compared with the range query. It is obvious that our ASR approach is more effective in




(a) Low density 100 objects




(b) Medium density 300 objects




(c) High density 500 objects

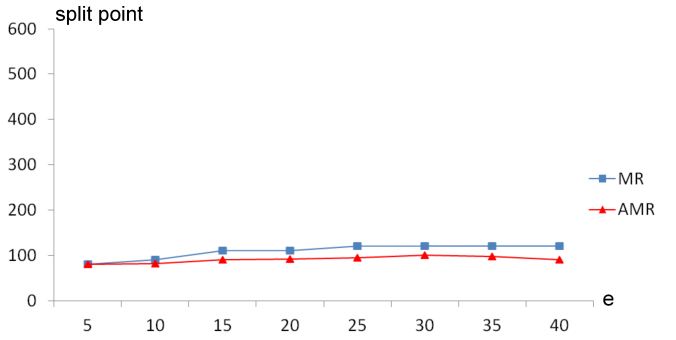
Figure 3.14: Number of false hits in ASR compared with range query (R)

a high density rather than low density environment, and it is less affected by the size of the range query.

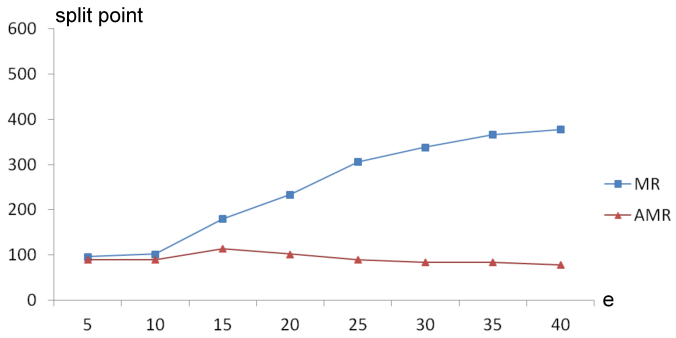
**Split Points in AMR**

Figure 3.15 compares the average number of split points between the moving range query and our proposed *Split Points Minimisation* approach. The experiment shows that when there are fewer objects (in a low density environment), then the number of split points is levelled in our proposed approach and is less than the number of split points in the moving range query approach. On the other hand, in medium and high density environments, the number of split points in our approach is completely different compared with the moving range query, especially when the size of the range search is increased. In our approach, the number of split points is decreased together with the increment of the interest objects. This occurs because the distance between the split points decreases. In addition, the number of split points is also decreased when the range search and/or the distance relative error are increased, due to the increased distance ratio error $\rho$. Figure 3.15(c) shows that in high density, when the range search is 40$km$, we obtain the smallest number of split points which is 10 times less than the moving range query. Overall, the *Split Points Minimisation* approach achieved the following: *i*) the number of split points is reduced by 20% in a low density environment, *ii*) the number of split points is reduced by 64% in a medium density environment, and *iii*) the number of split points is reduced by 80% in a high density environment.

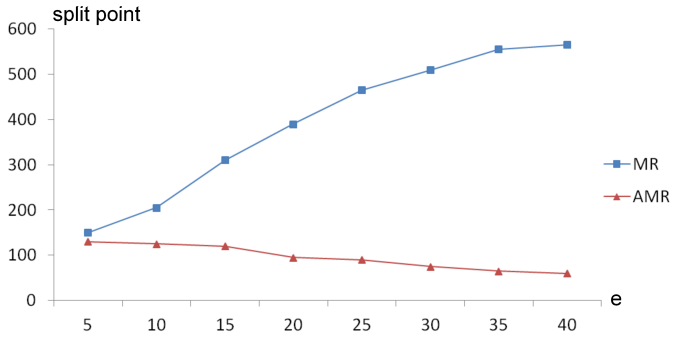
Figure 3.16 shows the average result of the experiments that examine the number of split points in the *Range Search Minimisation* and compare it with the moving range query. Similarly, we use 200 random queries on different object densities (low, medium, and high) in ASR experiments. We find that, the size of the query is not effect the number of split points in the low density environment. Also, the number of the split points is almost same in all environment when




(a) Low density 100 objects




(b) Medium density 300 objects




(c) High density 500 objects

Figure 3.15: Approximate splitting point in AMR based on Euclidean distance compared with moving range query (MR)

the size of the query is small. Therefore, our approach of *Range Search Minimisation* is affected more by a high density environment and a large range size. On average, 23% of the critical objects are excluded and false hits are half compared with the moving range query.

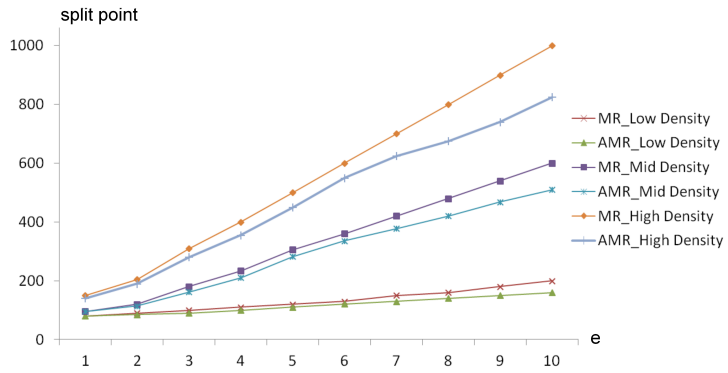
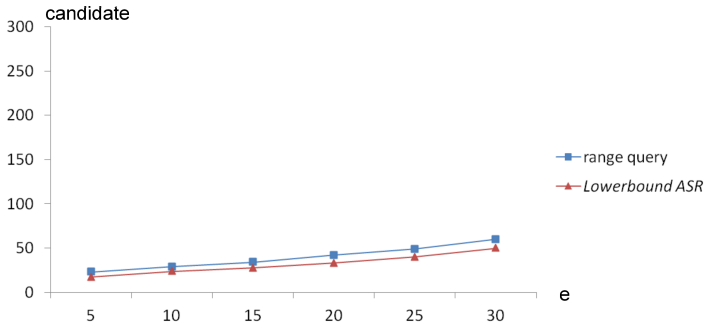



Figure 3.16: Number of split points in AMR and moving range query (MR)

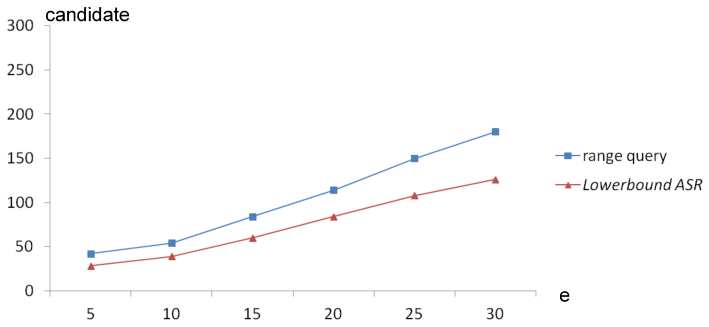
### Candidate Objects and False Hits in *Lowerbound* ASR

Figure 3.17 depicts the performance of our algorithm *Lowerbound* ASR with a diverge range search from 5*km* to 30*km*. The effect of increasing the range search $e$ on the number of candidate objects is similar between the range query and *Lowerbound* ASR. However, the candidate objects in *Lowerbound* ASR are less than the candidate objects in the range query. Also, on average, the false hits in the range query are approximately greater than the false hits in *Lowerbound* ASR by a third. The false hits increase with the size of the range search $e$, and also with the density environments.

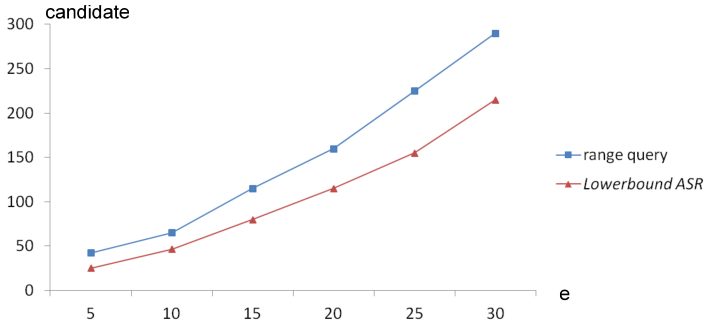
Figure 3.18 depicts the percentage of false hits which are 17%, 35% and 38% in low, medium and high density respectively. This outcome is less compared to the range query. In respect to these results, the performance of our approach *Lowerbound* ASR is more efficient in high density environments than low density, and it is slightly affected by the size of the range search.




(a) Low density 100 objects




(b) Medium density 300 objects



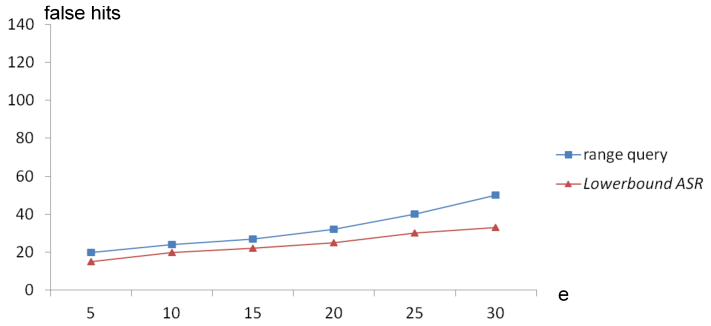

(c) High density 500 objects

Figure 3.17: Filter step

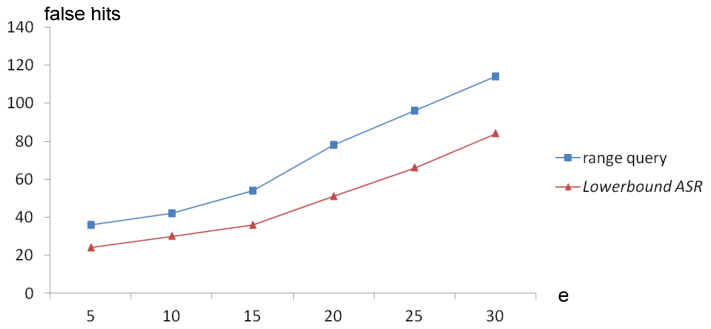
### 3.6.2 Query based on Road Network Distance

#### Objects of Interest and False Hits in ARER

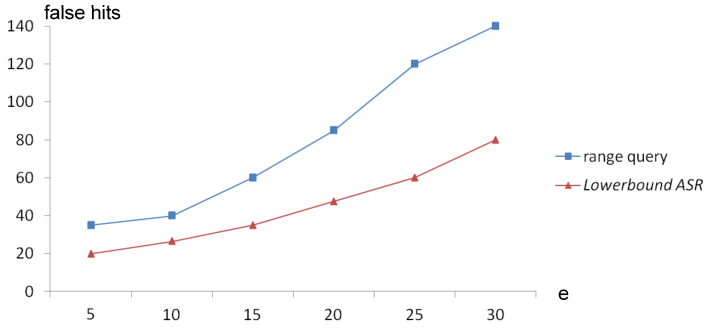
In RER, candidate objects within the Euclidean range $e$ are retrieved first, and then the network is expanded. While, in ARER candidate objects within the approximate Euclidean range $(e \times (1-\gamma))$ are retrieved first, and then the




(a) Low density 100 objects




(b) Medium density 300 objects




(c) High density 500 objects

Figure 3.18: False hits in Euclidean distance

network is expanded. Thus, the number of false hits in the ARER are less than in RER; RER retrieves more false hits in Euclidean distance than in ARER, and consequently leads to more R-tree searches.

Figure 3.19 shows a comparison of the accuracy, number of candidates, and number of false hits between ARER and RER. The accuracy of the result in ARER is more than 90%, with the knowledge that the retrieved candidates that

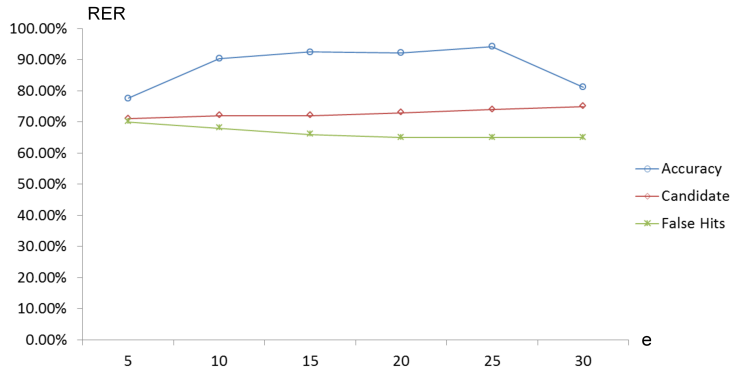



Figure 3.19: Approximate Range Euclidean Restriction ARER

we examined are about 70%. Consequently, the false misses in ARER are less than 10%, because most of the candidates that have been excluded are located beyond $(e \times (1-\gamma))$ of the query point. These false misses are not very important to the user that invokes the query, because they are far from the query point and they represent a quantity that can be neglected.

On the other hand, the search time of ARER is significantly less than the search time of RER, since many MBRs have been excluded from the search in the Euclidean range search and the number of candidates that have been processed in the network refinement step is less in ARER.

### Interest Objects and False Hits in ARNE

RNE, first, expands the network by distance $e$, and then performs the query on R-tree data for the actual result. While ARNE, first, expands the network distance to $e$ distance if the start node of the current segment is less or equal to $e \times (1-\gamma)$, otherwise the expansion will be between $e \times (1-\gamma)$ and $e$ from the query point $q$. Then, it performs the query on R-tree data for the actual result.

Both RNE and ARNE should find the qualified segment first. We find out that the number of pre-computations to locate the qualifying segment has dropped significantly when ARNE is used. This is because the segments that fall

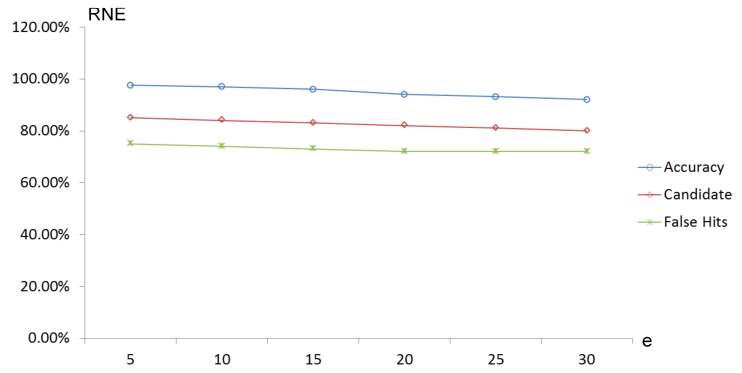



Figure 3.20: Approximate Range Network Expansion ARNE

after $e \times (1-\gamma)$ are excluded from the expansion. ARNE is similar to ARER were the false misses are far from the query point (i.e., $e \times (1-\gamma)$ network distance is beyond the $q$) and they represent a quantity that can be neglected.

Figure 3.20 illustrates a comparison of the accuracy, number of candidates, and number of false hits between ARNE and RNE.

### Split Points in AMR

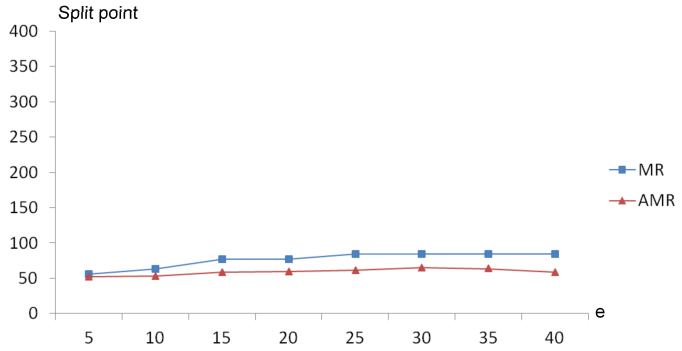
There are two main factors that are inversely proportional with the number of split points: the density environment and the radius of the query. Figure 3.21 compares the average number of split points between the moving range query based on road network and our proposed approximate moving range query (*Split Points Minimisation* approach) based on road network. The experiment shows that when the objects are not many (in a low density environment), then the number of split points is levelled in our proposed approach and is less than the number of split points in the moving range query approach. On the other hand, in medium and high density environments, the number of split points in our approach is completely different compared with the moving range query, especially when the size of the range search is increased.

In our approach, the number of split points decreases together with increments of the interest objects. This occurs because the distance between the split points decreases. In addition, the number of split points also decreases when the range search and/or the distance relative error are increased, due to the increased distance ratio error $\rho$. Figure 3.21(c) shows that in high density, when the range search is 40*km*, we obtain the smallest number of split points which is 10 times less than the moving range query. Overall, the *Split Points Minimisation* approach achieved the following: *i*) the number of split points is reduced by up to 30% in a low density environment, *ii*) the number of split points is reduced by up to 80% in a medium density environment, and *iii*) the number of split points is reduced by up to 90% in a high density environment.

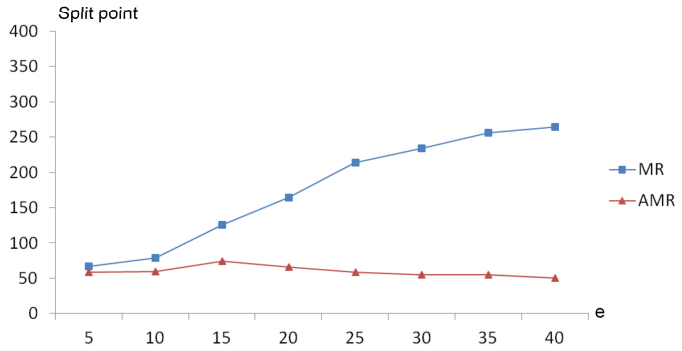
## 3.7 Summary

In this chapter, we propose four new approaches namely, *approximate static range query* ASR and *approximate moving range query* AMR which are based on Euclidean distance, and *approximate static range query* and *approximate moving range query* AMR which are based on network distance. We summarise our work as follows:

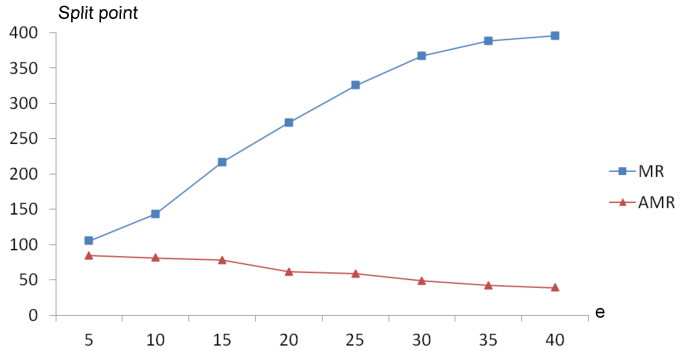
1. We propose a new query type, ASR, to determine the location of interest objects around a static point. The aim of this query is to obtain results quickly by reducing the number of false hits, thereby reducing the number of input/output operations. Also, this query gives the users alternative options which reduces the likelihood of making additional searches. Thus, our approach provides the search result with a high quality guarantee, i.e. all objects near the query will be included in the result.




(a) low density 100 objects




(b) mid density 300 objects




(c) high density 500 objects

Figure 3.21: Approximate splitting point in AMR based on road network compared with moving range query (MR)

2. We propose another query type, AMR based on Euclidean and network distance, to determine the locations of interest objects while the query moves on a pre-defined path. Two methods are proposed to obtain the approximate optimal search result: *Range Search Minimisation* and *Split*

*Points Minimisation*. The aim of this new query is to reduce the number of critical objects that enter the range search and leave it in a very short time. This will give the user enough time to make a decision. Moreover, this query reduces the number of split points, reducing the number of false hits, and also reducing the number of communications between the mobile device and the database server. The results of these experiments show that our proposed algorithms are successful in term of reducing the number of split points and false hits. Also, the approximate results were of a high quality compared to the exact results.

3. We present two comprehensive approaches for the range query based on road network, where the approximate model is used for its high quality results compared to the exact one. Our approaches, Approximate Range Euclidean Restriction (ARER) and Approximate Range Network Expansion (ARNE), give a better performance in terms of search time and search accuracy.

   In ARER, we determine the location of interest objects around a static point in the road network. ARER obtains the result in a short time by reducing the number of pre-computations when some of the MBRs are excluded from the search. Also, ARER reduces the number of false hits, and thus the number of I/Os and communications between the mobile device and the database server are also reduced. In addition, this approach neglects any false misses since they are located out of the approximate range and considerably away from the query point. Therefore, the technique provides the answer with a guaranteed high quality, that is, all objects near the query will be included in the result.

Our second approach, ARNE, determines the locations of interest objects in SNDB within a pre-defined path. The aim of this query is to reduce the number of nodes that should be visited while searching for the qualified segments. This reduces the number of computations. In ARNE, we exclude any nodes that fall between the *lowerbound* and the range search, thus, the results of our experiment show that ARNE is functioning efficiently in term of reducing the number of pre-computations and false hits.

In terms of accuracy, the main difference between ARER and ARNE is that ARER accuracy increases when the range search expands, whereas the accuracy of ARNE decreases. This is due to the nature of the search which is restricted in ARER and expanded in ARNE. However, both ARER and ARNE are guaranteed to include all objects within the network distance $e \times (1-\gamma))$ from $q$.

# Chapter 4

## Safe Region in Moving Range Query



**Publications and Submissions:**

1. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), On finding safe regions for moving range queries. In *Mathematical and Computer Modelling*. Elsevier, 58(5-6), pp.1449-1458.

2. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), Dynamic safe regions for moving range queries in mobile navigation. In *International Journal of Ad Hoc and Ubiquitous Computing*. (Accepted)

# 4

# Safe Region in Moving Range Query

The moving range query represents one of the most important types of processing for spatial and geographical information systems. The moving range query can be defined as: given a set of special objects, a query point and a range (radius), find all objects of interest within the radius of the query while the query moves. In moving range query, the queries are assumed to be constantly moving. Many user's queries such as those related to traffic control, multimedia search engines, on-line search engines, Geographic Information Systems (GIS) and wireless sensor networks require an information system with an efficient implementation of moving range queries (Lee et al., 2013; Price, 2012; Ahmed & Kanhere, 2012). An example of moving queries using Location-Based Services (LBS) is a car driver who wants to find all petrol stations within a radius of ten kilometres from his/her location (Wu & Hsieh, 2012; Jayaputera & Taniar, 2005). Another example is a pedestrian searching for a takeaway food outlet within a 500 metre range while walking in the city.

Minimising the frequent updates of the query location and keeping low costs while monitoring the moving query are the two main challenges researchers have to face. In this chapter, we will address these challenges and present new efficient methods using the safe region concept, called Range Safe Region.

The rest of the chapter is organised as follows. Section 4.1 gives the motivation of our proposal. Section 4.2 describes our proposed methods. Then, Section 4.3 presents a calculation for the proposed safe range area. Section 4.4 presents the algorithms used. Performance experiments are covered in Section 4.5. Finally, Section 4.6 summaries the chapter.

## 4.1 Motivation

Mobile navigation is one of the most common applications of mobile information services (Berg et al., 2008; Alamri, Taniar, & Safar, 2013). It is a technology that helps users navigate crowded roads while moving, guiding them to the best route, and answering their queries. There is also an increased demand for this technology due to the proliferation of mobile mapping and Internet search-capable devices. For mobile searches, different types of spatial moving queries have been used; such as moving range query, moving nearest neighbour query and moving joining query. The exponential nature of the problem search space means that the search process is computationally intensive. Also, the continuous updating of the query's location can be prohibitively expensive, depending on the nature of the query and the data properties (Alamri, Taniar, Safar, & Al-Khalidi, 2013; Taniar et al., 2008). The relatively high complexity of query processing has led researchers to attempt to solve the problem by introducing the concept of the safe region (Cheema et al., 2011; Cho et al., 2013).

The safe region is an area where the set of objects of interest does not change as long as the query remains inside it. The aim of the safe region is to reduce the number of query location updates by reducing the number of queries to the server. The main challenge is how to keep the query result up-to-date while the user is moving and how to reduce to the minimum the server's monitoring for the user. The Voronoi diagram is the traditional method of calculating safe regions (Okabe et al., 2000). Using the Voronoi diagram as a safe region has some major limitations. For example, updating the Voronoi diagram continuously is very expensive, in addition to the fact that the Voronoi diagram cannot deal efficiently with the updating of objects in the underlying dataset. Furthermore, the Voronoi diagram as a safe region suits only moving nearest neighbour queries, not moving range queries. In a moving range query, the result of objects of interest may change even if the query stays in the same Voronoi cell. In response to the Voronoi cell-based limitations and the limitations of other methods that generate safe regions, referred to in Section 2.4, we have introduced three new types of safe regions called: Basic, Enhanced, and Extended. Monte-Carlo simulation is used to calculate the total area of the Extended Safe Region due to its irregular shape.

Our approaches have the following specifications: First, we introduce an effective approach to construct a safe region that deals efficiently with updating query results, where the query contacts the server only if it leaves the safe zone (safe region). Second, our approach supports the concept of random query moves. Finally, we compare the initial Basic, Enhanced and Extended Safe Regions together, then we introduce a comparison between continuous Basic, Enhanced and Extended methods when the query is moving. The comparison uses the total area of the safe region(s) as the performance measure. Our

research presented in this chapter was published in (Al-Khalidi et al., 2013b; AL-Khalidi, Taniar, Betts, & Alamri, 2013).

## 4.2 Range Safe Region

The problem of when to update a query location is a major concern for all mobile communication systems and location dependent applications (Ilarri et al., 2012). Our proposed techniques are intended to minimise the costs associated with the communication of moving range queries in mobile navigation and also to support the concept of random query moves. To minimise the downlink messages of location searches, which are caused by query movements, adaptive safe regions can be utilised. Low communication costs and higher scalability, which are the main advantages of the aforementioned solutions, are improved with our technique. In this section, we construct a safe region formed by the intersection of range objects which allows the query inside it to move freely. In this safe region, the query needs to issue a location update only when it leaves that safe region. With our methods, the server will send only the updated result and the new safe region to the query.

In general, some types of safe regions need to report their query locations to the server after every $t$ time units or $d$ distance units. Assuming that $t$ or $d$ units are safe regions will lead to an incorrect result (Yung et al., 2012). In addition, the main limitation regarding the search is that the existing approaches use rectangular safe regions only (Zhang et al., 2003), and they are not applicable to the moving circular range queries. A recent search (Cheema et al., 2011) introduced a circular safe region for a moving query. This method did not calculate the area, which means the server is always in a stand by situation, expecting contact from the query. In our approach we use the Monte-Carlo

method to calculate the area of the safe region area due to its irregularity. The following subsections will describe our approaches.

### 4.2.1 Basic Safe Region

Our monitoring framework will solve the problem of frequent updating of the query location by taking a systematic approach. Whereby the server is aware of the location of the moving query and location updates occur only when the query passes out of its specified safe region.

The safe region of the query is computed at the server side based on the closest point, inside/outside the range query, to the border. The computed safe region is sent to the moving query by the server. The query then sends its location to the server when its location is outside that safe region.

Consider that the first nearest object is outside the boundary of the moving query and far away from the range query. The query has to move this distance before the object becomes within the range search (boundary) of this query. Figure 4.1 illustrates this: $p4$ is the nearest object to the range boundary which falls outside it and $N_d$ is the distance between $p4$ and the boundary of the query. At this point, the query should move the $N_d$ distance until $p4$ becomes within the range query. Consider that $p3$ is the furthest object of interest of the moving query within the boundary. In Figure 4.1, $F_d$ is the distance between $p3$ and the boundary of the moving query. In this case, the query also has to move this far distance ($F_d$) before this object moves out of the range search of this query.

Let $N_d$ be the minimum distance between the closest object to the query outside the range boundary: we call this object *firstclose* ($pt$), and the range boundary itself ($N_d = MinDist(pt,q) - e$). Accordingly, the moving query should

move the distance of ($N_d$) in order for this object to come within the range query. Let $F_d$ be the minimum distance between the furthest object from the query within the range boundary: we call this object *farclose* ($pr$), and the range boundary itself ($F_d = e - MinDist(pr,q)$). Accordingly, the moving query should move the distance ($F_d$) in order for this object to be outside the range search query. The safe region will then be a circular area around the query, the centre of which is the query's initiated location and its radius is the smallest distance between $N_d$ and $F_d$ notated by ($SmDist(N_d, F_d)$).

The server does not need to check the location of the query as long as it does not move outside the safe region. This means that while the query does not go further than the *SmDist* from its original location, there is no need to check its location. Figure 4.1 shows examples of the *Basic Safe Region*, where all objects are indexed in R-tree. $P3$ represents the furthest object of interest within the range of the moving query $q$ (*farclose*), and $p4$ represents the closest object outside the boundary of the range query (*firstclose*). Let $e'$ represents the radius of the safe region:

$$e' = SmDist(N_d, F_d) = \begin{cases} F_d & \text{if } F_d < N_d \\ N_d & \text{otherwise.} \end{cases} \tag{4.1}$$

Using this scheme, $q$ can move in any direction with a distance of $e'$ from its original location without affecting its result; in other words, $q$ does not need to update its location while moving within radius $e'$ from its original location.

The current result of the query is guaranteed to remain valid as long as the query remains inside its respective safe regions. The query will inform the server of its new location when it leaves its safe region and the server will

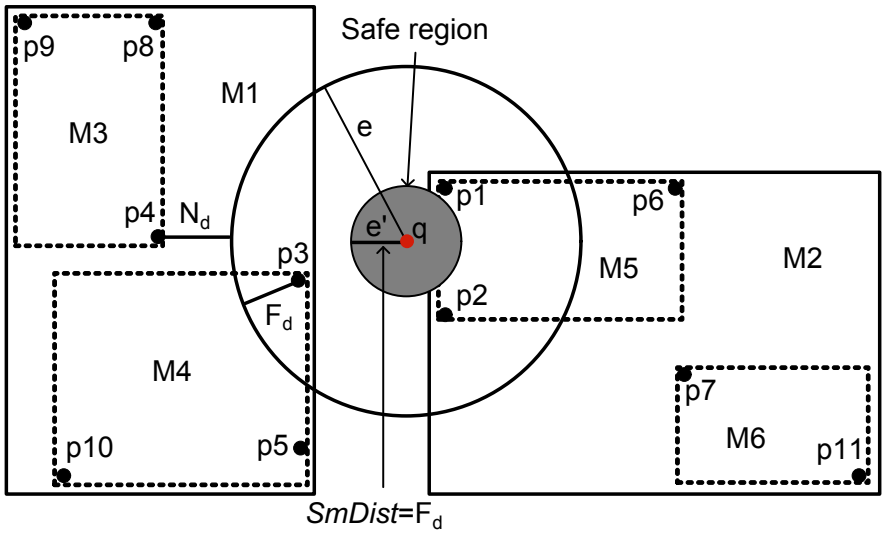



Figure 4.1: Basic Safe Region

pre-compute a new safe region for the query and send this new region to the moving query.

The query is aware of its safe region and issues location updates to the server only when it moves out of the region (this is called a source-initiated update) (Cheema et al., 2012). After the server receives this update, it finds and incrementally re-evaluates the affected objects and computes a new safe region for the moving query. The server will then send the new safe region to the query together with the set of updated objects of interest.

### 4.2.2 Enhanced Safe Region

Enhanced Safe Region represents an intermediate case between the Basic and the Extended Safe Regions. This method constructs the safe region more easily by forming a large enough safe region within which the query can move freely. The Basic Safe Region assumes that the query will be outside its safe region when it moves by $e'$ from its initiated location. However, the fastest way that the query will leave its safe region is when it moves to the exact direction of

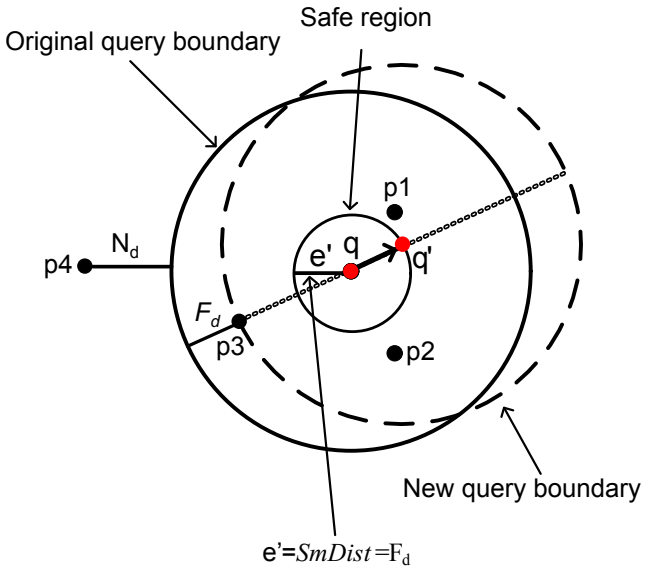



Figure 4.2: Query moves in opposite direction

the closest object to the border (if the object is outside the boundary of the query for example, toward $p4$ in Figure 4.2) or to the exact opposite direction of the closest object to the border (if the object is inside the boundary of the query for example, away from $p3$ in Figure 4.2, if we assume $p3$ is the closest object to the boundary).

The Basic Safe Region can be enhanced by taking into account the closest two objects to the boundary of the query. Let $e$ be the radius of the range query, $d_{in}$ the distance between the query and $p1$, and $d_{out}$ the distance between the query and $p2$. Figure 4.3 shows that $p1$ is the closest object and $p2$ is the second closest object to the *boundary* of the query respectively. The shortest path the query will only follow to change its objects of interest is when it moves exactly to the opposite direction of $p1$, after moving the distance of ($d'_{in}$). However, if the query moves to the exact direction of $p2$, which is the second closest object to the range query's border, then the query will change its objects of interest when it moves by distance ($d'_{out}$).

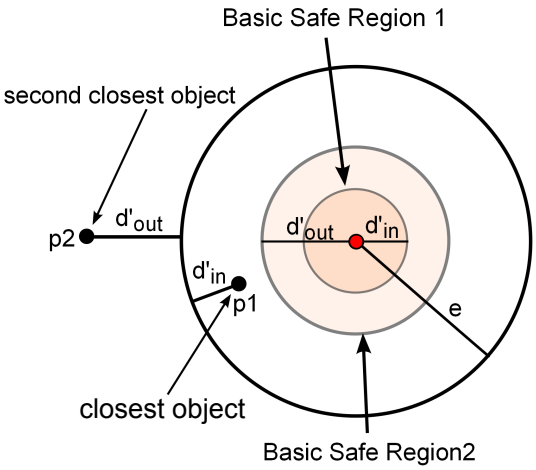



Figure 4.3: Two closest objects to the border of the query

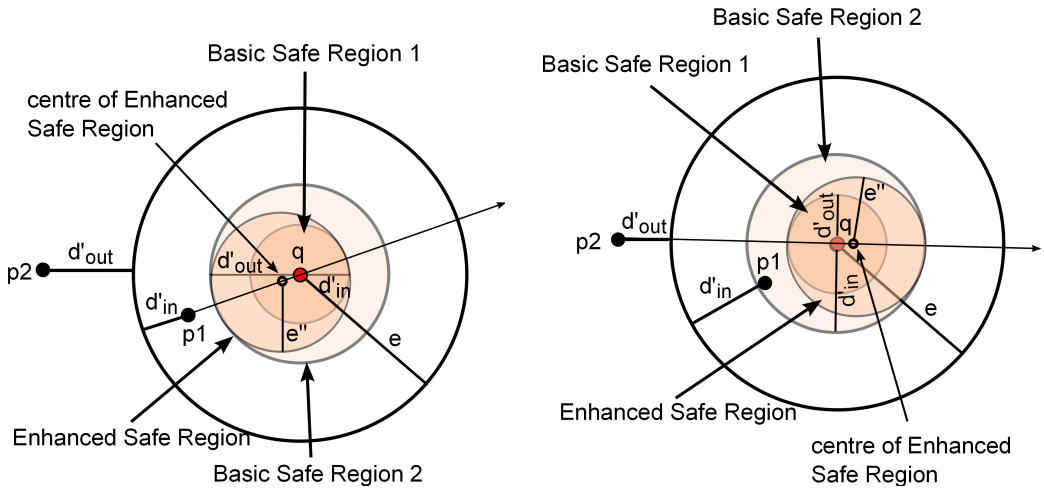
$$d'_{in} = e - d_{in} \tag{4.2}$$

$$d'_{out} = d_{out} - e \tag{4.3}$$

As shown in Figure 4.3, the Basic Safe Region (1 and 2) can be merged together to obtain a new safe region. The radius of the new safe region will be the mean distance between the $d'_{in}$ and $d'_{out}$,

$$e'' = \frac{d'_{in} + d'_{out}}{2} \tag{4.4}$$

In the Basic Safe Region, the query represents the centre of the safe region, but in the Enhanced Safe Region, the centre of the safe region is not the query. The Enhanced Safe Region has a circular shape and by using Equation 4.4, we can calculate its radius. However, the location of its centre is still unknown. By observation, we find that when a line is drawn from the closest object to the border through the query, the centre of the Enhanced Safe Region will be




(a) Closest object inside the range (b) Closest object outside the range

Figure 4.4: Enhanced Safe Region

allocated on that line. Also, the border of Enhanced Safe Region should go through the intersection of the Basic Safe Region and the line.

Figure 4.4 shows two scenarios for constructing the Enhanced Safe Region. Figure 4.4(a) shows when the closest object to the border is inside the range query, while Figure 4.4(b) shows the closest object to the border is outside the range query. In both scenarios, the radius of the Enhanced Safe Region is calculated using the Equation 4.4. The centre of the safe region will be located on the line that crosses the query, starting from the closest object to the border. The centre of the Enhanced Safe Region will be on that line and the border of the Enhanced Safe Region will intersect with the intersection point between the Basic Safe Region 1 and the line.

### 4.2.3 Extended Safe Region

The approach presented in Section 4.2.1, assumes that the query will be outside its safe region when it moves by *SmDist* from its initiated location. However,

the fastest way that the query will leave its safe region is when it moves to the exact same/opposite direction of the *firstclose/farclose* object.

**Lemma 4.1.** *The shortest path the query will follow in order to change its set of interest objects is when it moves in exactly the same/opposite direction to the firstclose/farclose.*

*Proof.* (by inspection) Since the firstclose/farclose object is the closest to the original query boundary then movement close/away to/from this object in a straight line, in the direction of the original query ($q$), gives the shortest path to exit the safe region. □

Figure 4.2 shows that moving query $q$ moves by $F_d$ from its initial location in the opposite direction of the object of interest $p3$. The new location of the moving query is $q'$. The new boundary around $q'$ signifies that $p3$ will be excluded from the set of the objects of interest with a guarantee that there will be no excluded objects and no added new objects to the set of the objects of interest.

**Lemma 4.2.** *An object will be excluded from the result of the moving query iff its distance to the query becomes greater than the radius of the query.*

*Proof.* Assume that there is only one object in the dataset $P = \{p1\}$. Also, assume that the minimum distance between this object $p1$ and the query $q$ ($MinDist(p1, q)$) is less than the radius $e$ of the $q$.

A 2-dimensional circular range query asks for the points from the data set ($P$) lying inside its range. Let $(q_x, q_y)$ be the coordinates of the query $q$, $e$ the radius of the range query and $(pi_x, pi_y)$ be the coordinates of object of interest $pi$, then

$$MinDist(pi,q) = \sqrt{(q_x - pi_x)^2 + (q_y - pi_y)^2}\ . \tag{4.5}$$

The location of the object $pi$ in terms of the range query will be determined according to $MinDist(pi,q)$ and the radius of the range. All the points on the boundary (circumference) of the range are fixed from the query point $q$, due to the circle definition,

$$MinDist(pi,q) - e \begin{cases} > 0 & pi \text{ outside boundary} \\ = 0 & pi \text{ on boundary} \\ < 0 & pi \text{ inside boundary.} \end{cases} \tag{4.6}$$

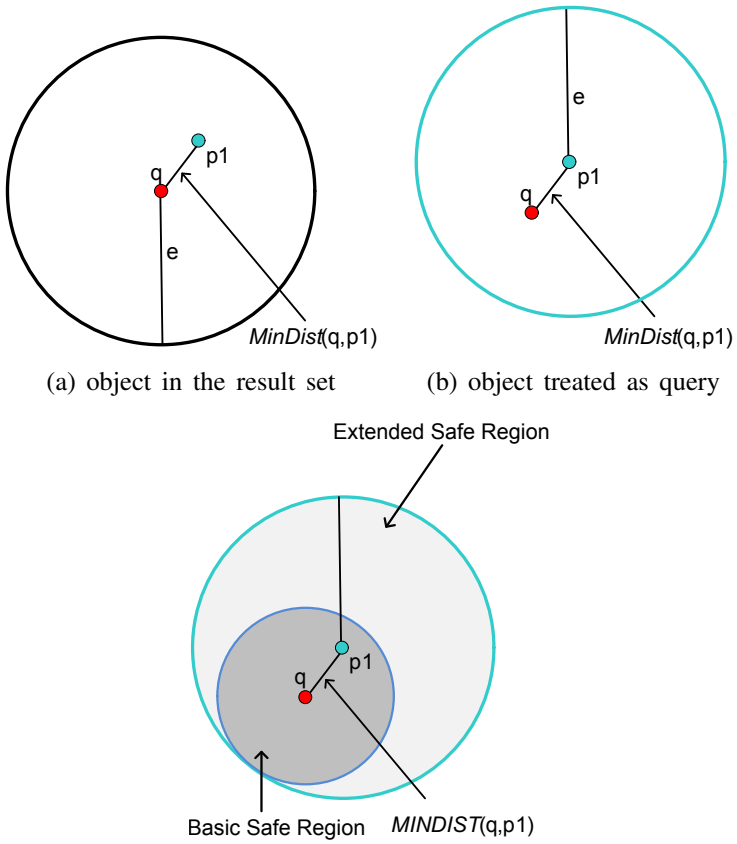
According to the definition of the range query, any object on the boundary will be within the result of the range. There will be two configurations to consider, depending on whether $pi$ lies inside the range search (including points on the boundary) or outside the range search. $pi$ lies inside the range if and only if:

$$(q_x - pi_x)^2 + (q_y - pi_y)^2 - e^2 \leq 0\ , \tag{4.7}$$

otherwise, $pi$ will lie outside the range and will not be within the result list. □

Figure 4.5(a) shows $p1$ is located inside the range of the moving query. Whenever a query moves, $p1$ will stay within the range of the query $q$ when $MinDist(p1,q)$ is not greater than $e$.

Treating the objects as a query, each object is surrounded by a circle equal to the range search of the query, see Figure 4.5(b). By Lemma 4.2, $p1$ will be in the result set since the query moves inside the range of the object $p1$. $p1$ will be excluded from the result when $q$ leaves the range of $p1$ ($MinDist(p1,q) > e$).




(a) object in the result set (b) object treated as query

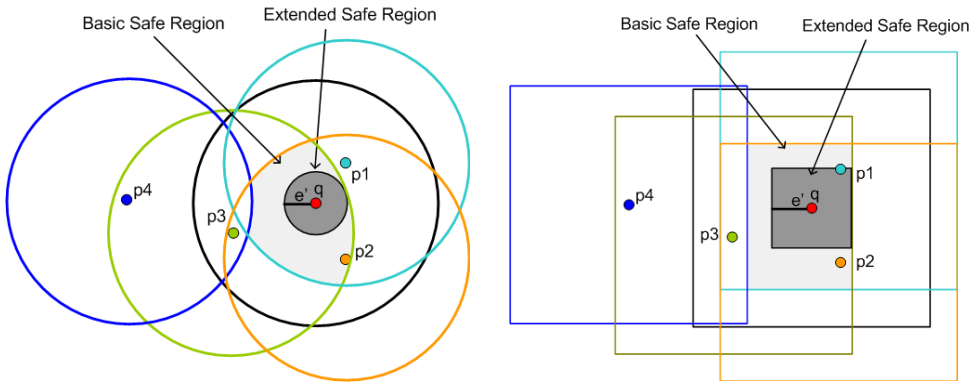
(c) Extended Safe Region

Figure 4.5: Treating objects as query

Consequently, in this approach, the safe region will represent the whole range of $p1$, while the safe region according to the previous approach (Basic Safe Region) will be the small circle, as shown in Figure 4.5(c).

Our approach presented in Section 4.2.2 can be extended. Each object (*inside* and *outside* the range boundary) will be surrounded by a boundary (circle or a rectangle depending on which type of range is used) range equal to the range search of the query, (see Figure 4.6). The intersection of these circles (or rectangles) will generate a polygon containing the query point. This polygon represents the Extended Safe Region.

Figure 4.6 shows an irregular/rectangle shape containing the query point $q$. This shape represents the Extended Safe Region. This new region is larger than




(a) circular range query (b) rectangular range query

Figure 4.6: Extended Safe Region (formed by overlapping objects)

the Basic and the Enhanced Safe Regions, giving the moving query more space in which to move without informing the server of its new location. By using this technique, the number of communications between the query(s) and the server(s) will be reduced as will be the number of location updates.

## 4.3 Calculating the Area of the Extended Safe Region

When the safe region is comprised of one or two objects, the calculation of its area is a straightforward geometrical problem. However, for more than two objects, the calculation of the safe region is complex as the overlapping regions may have highly irregular forms. For this reason, we propose to calculate these areas using Monte-Carlo simulation. Consider a set $RP = \{Rp1, Rp2, ..., Rpn\}$ of $n$ circles, whose centres are $\{p1, p2, ..., pn\}$ and radius is $e$. The circles in $RP$ may partially overlap.

Figure 4.7 shows a variety of safe regions formed by five objects, $P = \{p1, p2, p3, p4, p5\}$ within a space $E$ and $RP = \{Rp1, Rp2, Rp3, Rp4, Rp5\}$. Each

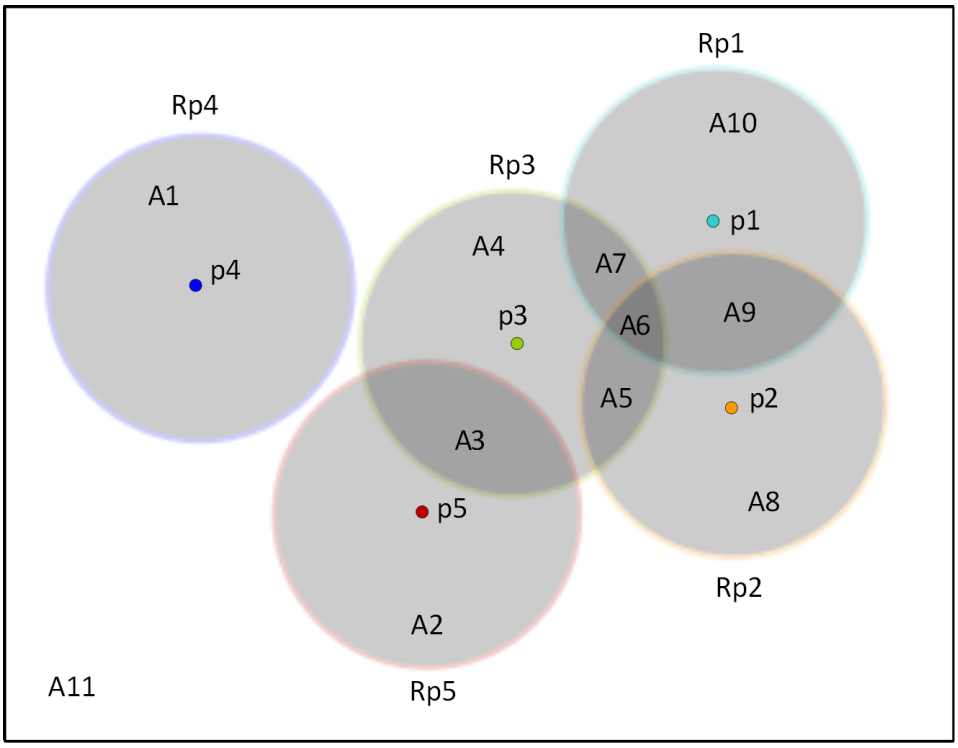



Figure 4.7: Types of Extended Safe Regions

$A$ represents a safe region, for example $A1$ represents one circular safe region while the others $(A2, A3, ..., A10)$ represent safe regions resulting from the overlap (intersection) of the range objects. Also, $A11$ represents a safe region but without any object in the query result list (no-object safe region).

When the query is within a range of one object and this range does not intersect with any other range, the whole range object will represent a safe region to query $q$.

$$MinDist(pi, q) \leq e \text{ and } Rpi \bigcap_{x \neq i} Rpx = \emptyset. \tag{4.8}$$

### 4.3.1 Calculating the Intersection of Two Circles

In many circumstances, the range of two objects may intersect and the query falls either within the range of one object $Rpi$ or within the intersection area of two objects $Rpi \cap Rpj$. In this situation

$$MinDist(pi,q) \leq e \text{ and } Rpi \bigcap_{x \neq i} Rpx \neq \emptyset \tag{4.9}$$

or

$$MinDist(pi,q) \leq e \text{ and } MinDist(pj,q) \leq e, i \neq j\ . \tag{4.10}$$

$A2$, $A3$ and $A4$, in Figure 4.7, represent the safe regions formed by the intersection of two circles (representing objects treated as queries). $A3$ is comprised of two half regions $A3'$ and $A3''$, because, in this case, the two circles are equal $A3' = A3''$. Their calculation is given by the following formula, where $2d$ is the distance between the objects and $e$ is the radius of the query, as depicted in Figure 4.8.

$$A3' = A3'' = e^2 arccos(\frac{d}{e}) - d\sqrt{e^2 - d^2}\ . \tag{4.11}$$

### 4.3.2 Using Monte-Carlo Simulation to Calculate Safe Region Area

For more than two objects, the safe region formed is potentially an irregular shape. As the area of these shapes cannot be calculated by a simple analytical expression, Monte-Carlo simulation (Ripley, 1987) is used instead to calculate their area. The area calculation requires that the queries be specified within a

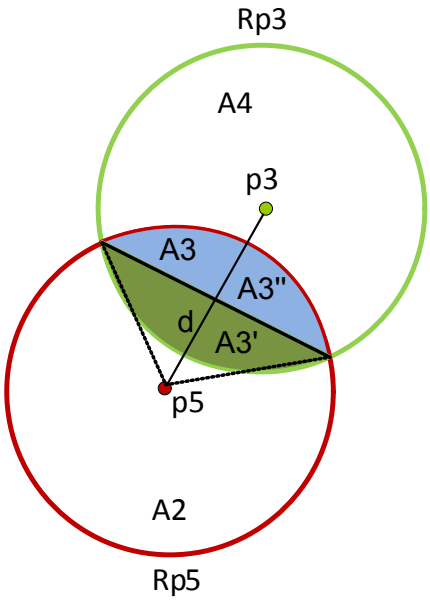



Figure 4.8: Area of intersection of two circles

bounding region of a known size. Multiple points within the bounding region are then generated at random, with the number falling inside the safe region being counted. The area of the safe region (SR) is then calculated as

$$\frac{sum\ of\ points\ in\ SR}{total\ random\ points} \times\ area\ of\ bounding\ region. \tag{4.12}$$

For the experiments reported in Section 4.5, the bounding region is determined as $100 \times 100 km^2$, with 100,000 random points being used to calculate the area of the safe region. As a guide to the accuracy of the method at these settings, a trial was conducted to calculate the area of the safe region for a single query having radius of $10km$. In this case, the expected area is $100\pi km^2$. 1000 test problems are generated having an average area of $314.0km^2$ and standard deviation 0.54, or approximately 2% of the area. Thus, it can be seen that even for a small safe region, the error of the method is quite small, diminishing proportionally as the size of the safe region increases. To illustrate this, a safe region of the size indicative of those reported in Section 4.5 is also calculated.

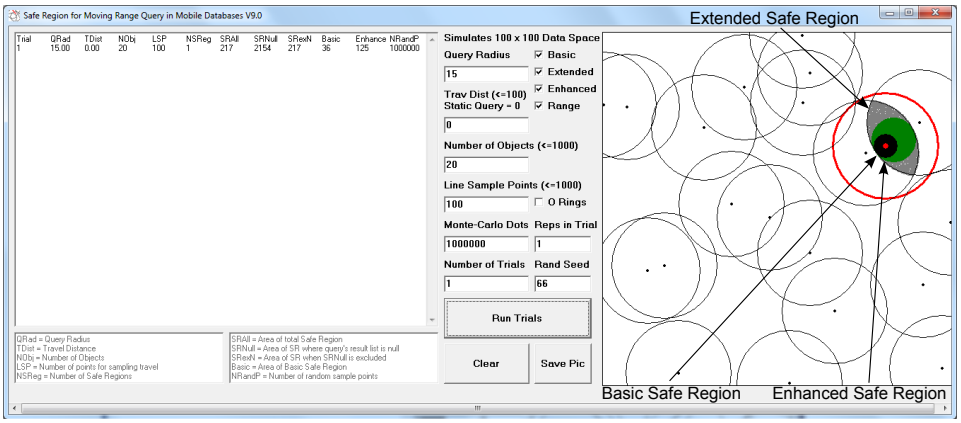



Figure 4.9: Demonstration software calculating the area of a safe region corresponding to a static query

Using a query radius of 4*kms*, Monte-Carlo simulation estimated the area of the region ($16\pi$ $km^2$) accurately to 3 decimal places as $50.274km^2$ having a standard deviation of 0.155, representing 0.3% of the calculated area.

Regarding the computational complexity of the Monte-Carlo simulation algorithm, running time is linear with respect to: *i*) the number of random points used to evaluate areas; *ii*) the number of query objects over the data space; and *iii*) the number of intervals at which a moving query is evaluated. Thus it is possible for a user to choose a desired level of accuracy or computational speed by varying these factors. The query radius has no effect on computational performance.

Figure 4.9 shows a screen shot of the simulation software used to calculate the area of the safe regions discussed in this paper. The region on the right hand side shows a query within a region of 20 objects. The black and grey areas correspond to the safe regions discussed later in the paper.

## 4.4 Algorithms of Safe Region

Full details of the Basic, Enhanced and Extended Safe Region algorithms are presented below.

### 4.4.1 Basic Safe Region Algorithm

Algorithm 4.1 is to find the Basic Safe Region where the safe-object list SOL, in the algorithm, is found first. This list would have all objects of interest and some objects just outside the range query. The safe-object list should be sorted in ascending order, and all objects within the distance $e$ from the $q$ will be moved to the result list (RL). The closest object to the border of the query will either be the last object in RL or the first object in what remains in SOL. Then, the query ($q$) will be surrounded by a circle with a radius equal to the distance between the border of the query and the closest object to this border. In this algorithm the search will be up to $2e$ from the query $q$

### 4.4.2 Enhanced Safe Region Algorithm

Algorithm 4.2 is to find the Enhanced Safe Region for a moving range query. In this algorithm all objects within $2e$ from the query $q$ should be found first (safe-object list SOL), then the safe region is calculated. After ordering SOL in ascending order, the closest two objects to the border of the query should be found. These two objects will be among a subset of SOL which has four objects (the closest two objects to the border of the query which are inside the range query and the closest two objects to the border of the query which are outside the range query). The average distance between the closest two objects to the border of the query will represent the radius of the enhanced safe region.

**Algorithm 4.1.** Basic Safe Region algorithm

```
/* q: query point and e: the Euclidean distance threshold */
/* Find objects of interest and a safe region q */
boundary = 2e
LIST RL, SOL
for each pi in data space do
  if MinDist(pi,q) > boundary then
    prune pi
  else
    if MinDist(pi,q) > e then
      boundary = MinDist(pi,q)
      SOL.add(pi)
    else
      RL.add(pi)
    end if
  end if
end for
sort(SOL), sort(RL)
Fd = e − MinDist(RL[length(RL)],q)
Nd = MinDist(SOL[1],q) − e
e' = min(Fd,Nd)
Surround the query (q) by a circle with radius e'
The area formed by a circle with radius e' is the Basic Safe Region of the
moving range query
```

### 4.4.3 Extended Safe Region Algorithm

Algorithm 4.3 calculates the Extended Safe Region for a moving range query. The set of the objects of interest is found first, after which the safe region is calculated. If $pi$ is the closest object to the query $q$ and $MinDist(q,pi) \leq e$, then any object $pj$ will be excluded from the calculation of the safe region if $MinDist(pi,pj) > 2e$ (AL-Khalidi, Taniar, Betts, & Alamri, 2013). Any object that falls above $2e$ from the closest object of interest ($p1$) to the query will not affect the safe region and will be pruned. The objects within distance $2e$ from $p1$ will be surrounded by the circular range of each object having radius $e$. The region formed by the overlap of each of these circular regions containing

**Algorithm 4.2.** Enhanced Safe Region algorithm

```
1: /* q: query point and e: the Euclidean distance threshold */
2: /* Find objects of interest and a safe region q */
3: LIST RL, SOL, closest
4: for each pi in data space do
5:    if MinDist(pi,q) > 2e then
6:       prune pi
7:    else
8:       if MinDist(pi,q) > e then
9:          SOL.add(pi)
10:      else
11:         RL.add(pi)
12:      end if
13:   end if
14: end for
15: sort(SOL), sort(RLt)
16: len = length(RL)
17: closest.add(e − RL[len − 1])
18: closest.add(e − RL[len])
19: closest.add(SOL[1] − e)
20: closest.add(SOL[2]) − e)
21: sort(closest)
22: e'' = (MinDist(closest[1],q) + MinDist(closest[2],q))/2
23: Surround the query (q) by a circle with radius e''
24: The area formed by a circle with radius e'' is the Enhanced Safe Region of
    the moving range query
```

$q$ forms the safe region of $q$ at this time. When the query moves outside the safe region, a new safe region should be allocated to the query with the updated result list. If there is no object within the range of the query, the search in this algorithm might cover all objects in the data space.

## 4.5 Experimental Results

Several experiments were conducted to evaluate our proposed algorithms. A synthetic dataset was used to represent the real world. We create three different density environments (low = 100 objects, medium = 300 objects and high = 500

**Algorithm 4.3.** Extended Safe Region algorithm

```
1: /* q: query point and e: the Euclidean distance threshold */
2: /* Find objects of interest and a safe region q */
3: LIST RL, SOL
4: for each pi in data space do
5:    if MinDist(pi,q) ≤ e then
6:       RL.add(pi)
7:    end if
8: end for
9: sort(RL)
10: /* for the remain objects in the data space */
11: if RL.isempty() then
12:    for each pi in data space do
13:       if MinDist(pi,RL[1]) > 2e then
14:          prune pi
15:       else
16:          SOL.add(pi)
17:       end if
18:    end for
19: else
20:    for each pi in data space do
21:       SOL.add(pi)
22:    end for
23: end if
24: sort(SOL)
25: for each pi in SOL do
26:    Surround object (pi) by a circle with radius e
27: end for
28: The area formed by overlapping regions containing q is the Extended Safe
    Region of the moving range query
29: Calculate the area of the safe region using Monte-Carlo Integration.
```

objects) in a data space measuring $100km \times 100km$. Then, we compare the Basic, Enhanced and Extended Safe Regions together for static and moving queries. We give three performance measures: the area of the safe region constructed using each technique, the number of objects needed to construct each safe region and the average distance the query can move in different density environments using the three safe regions. We randomly generate 1000 random queries in low, medium, and high density environments, and calculate the area of the safe

region. We also calculate the current safe region of the query and also the safe regions the query would pass through as it moves. The densities could be considered as representing queries for different services in a city. For example: hospitals (low), post offices (medium), and restaurants (high).

Because the number and size of safe regions resulting from a query are dependent on the density of the objects (appearing in a map for example) and the radius of the query, these factors were varied in the trials that follow. Two performance measures were used to evaluate the extended safe region. First, a comparison was made between the average size of the safe region and the whole size of the data space. Secondly, we counted the number of safe regions that are crossed when the query moves within the data space. The purpose of these experiments is to show: *i*) the accuracy of our simulation method; *ii*) the size of the safe region and the impact of different factors such as: object density; range query distance; and whether the query is moving or static on the size of the safe region; and *iii*) the number of objects needed to construct the Extended Safe Region. Figure 4.10 shows an example of the different safe regions generated by the simulation and the safe regions that the moving query will cross while moving in this path. The source of our implementations can be downloaded from the following URL: `https://www.dropbox.com/sh/azy7xqjpcu2387t/AAAceaGT10BPcQXQt86-hhl8a`.

### 4.5.1 Accuracy when Using Monte-Carlo Simulation to Calculate Safe Region Area

A safe region was constructed using two overlapping object ranges. The areas were calculated using the exact method illustrated in Section 4.3.1, and Monte-Carlo simulation method described in Section 4.3.2. We calculate this area using

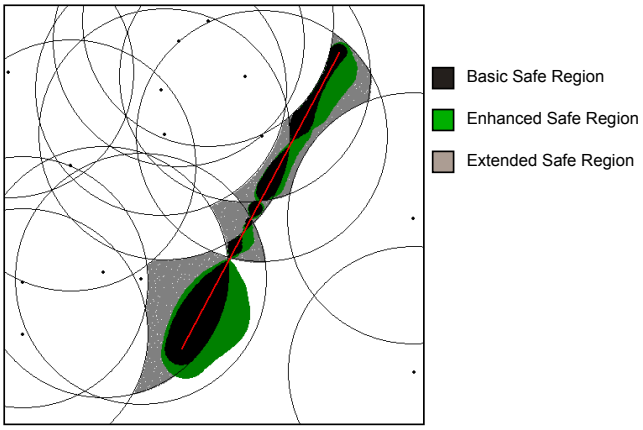



Figure 4.10: Example of safe regions (Basic, Enhanced and Extended)

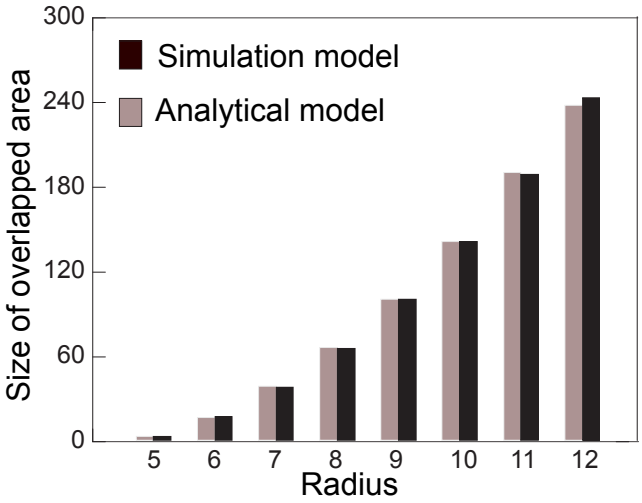



Figure 4.11: Simulation model *vs.* analytical model

query radii 5, 6, ..., 12$km$. The distance between the objects was fixed at 9 $km$ to ensure that query ranges would not overlap. The safe regions were calculated 100 times using Monte-Carlo simulation for each query radius, and the average of each radius was established and compared with the precise result to illustrate the accuracy of our simulation method. Figure 4.11 shows that the safe region area calculated using Monte-Carlo simulation is very close to the exact areas obtained using Equation 4.11.

### 4.5.2 Initial Safe Region

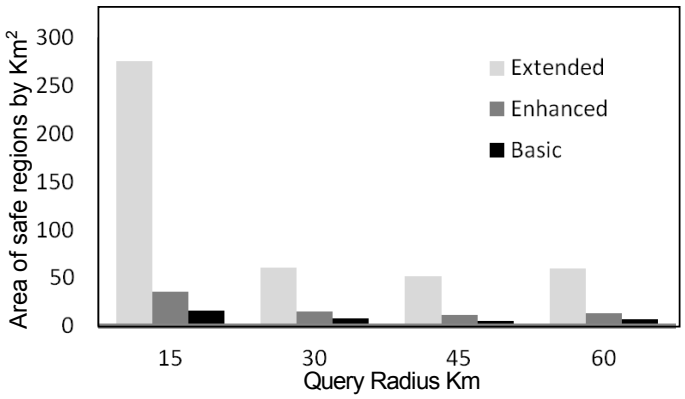
Figure 4.12 shows a comparison of the three types of safe regions: Basic, Enhanced and Extended, in three different environments (low (Figure 4.12(a)), medium (Figure 4.12(b)) and high (Figure 4.12(c))), with a radius of range search that varies from 15$km$ to 60$km$ and a static query. We found that, the size of the Enhanced Safe Region is more than twice the size of the Basic Safe Region in all of the environments, while it is four times smaller than the Extended Safe Region. In all environments, regardless of density, when the range search is small the majority of the safe region is composed of the area containing no objects.

### 4.5.3 Continuous Safe Region

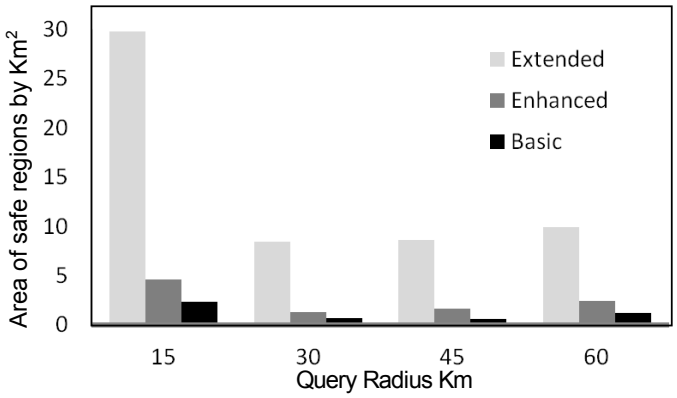
Figure 4.13 shows the total areas of a query moving 100$km$. The experiments show that varying the number of objects in the data space has more impact than varying the range search, especially in medium and high density environments. The effects of varying the range can be noticed only when the radius is small because this is the region with no nearby objects. When the query is moving, the area of the Enhanced Safe Region is approximately twice the size of the Basic Safe Region and less than three times the Extended Safe Region.

### 4.5.4 Constructing the Safe Region

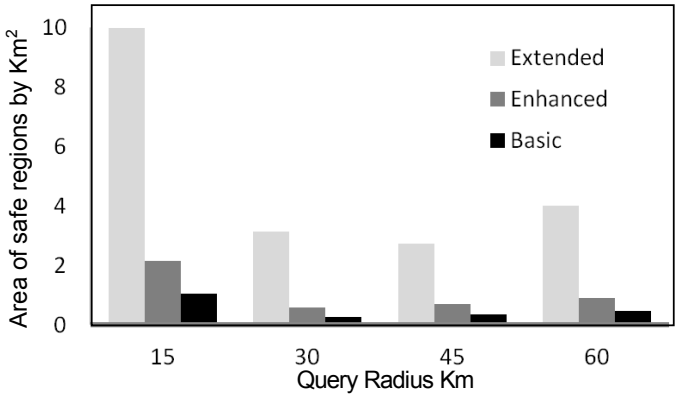
The closest object and the closest two objects to the border of the moving range query will be considered when constructing the Basic and the Enhanced Safe Regions respectively. However, when constructing the Extended Safe Region, many objects will be considered. Whenever the number of the objects increases around the query, the number of objects needed to construct the safe region will




(a) Low Density Environment



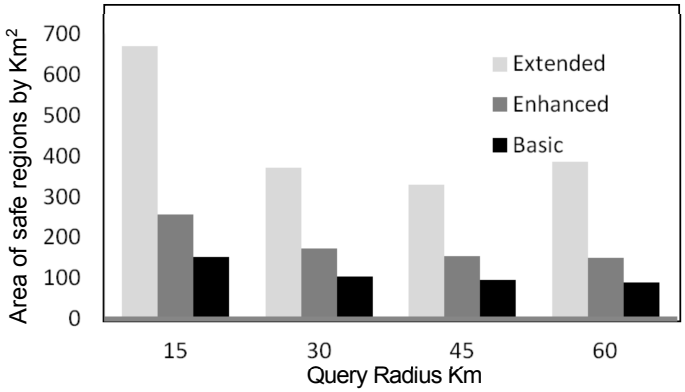

(b) Medium Density Environment



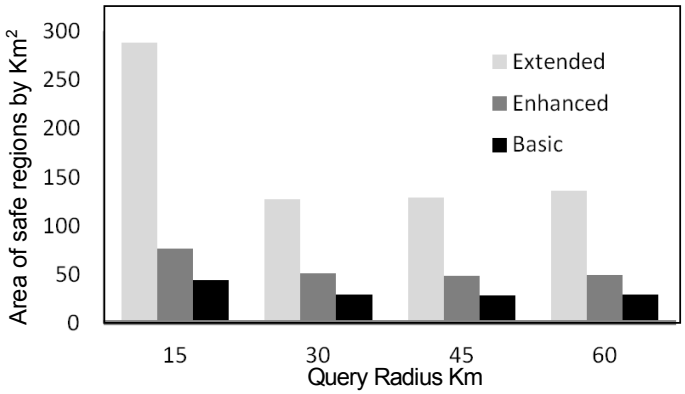

(c) High Density Environment

Figure 4.12: Initial safe regions in different environments

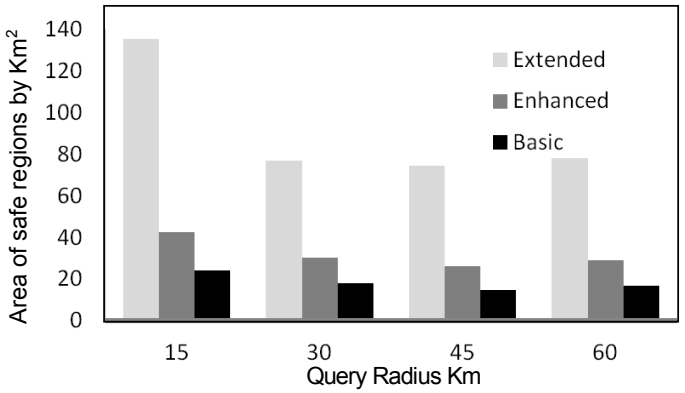
increase too. For example, Figure 4.14 shows when the range query is small, almost all the objects in the data space will be used to create the Extended Safe Region.




(a) Low Density Environment




(b) Medium Density Environment




(c) High Density Environment

Figure 4.13: Total area of safe region crossed by moving query

## 4.6 Summary

In this chapter, we propose a continuous range safe region method for mobile navigation with moving queries. Our contributions can be summarised as follows: First, we have proposed new techniques to construct a safe region, termed

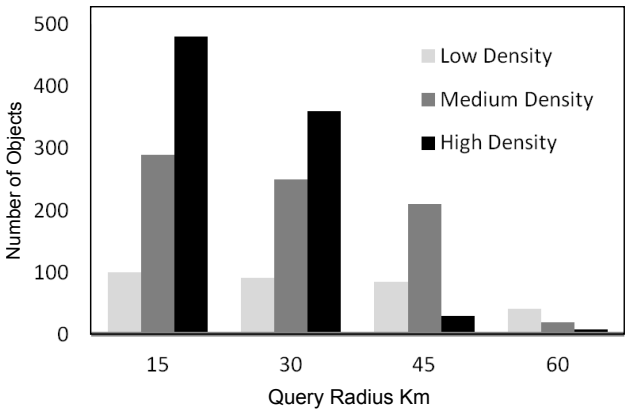



Figure 4.14: Number of objects needed to construct the Extended Safe Region

the range safe region. The aim of these techniques is to avoid any communication between query and server while the query moves within the specified safe region. Second, we have extended the Basic and Enhanced Safe Regions to reduce the amount of supplementary communication. Using these techniques reduces the need for continuous monitoring of the query, and eliminates the need for the user to follow a defined path. We use a discrete-event simulation method with a high degree of accuracy to calculate the area of the safe regions that have been formed by overlapping range objects. Finally, we compare the Basic, Enhanced and Extended Safe Regions when the query is static and when it is moving. The experimental results show that our method reduces the amount of query monitoring by the server, and then are no communication costs, while the query moves within its safe region. The new method also allows a user to move within the safe region without revealing their location, thus preserving their privacy.

# Chapter 5

# Monitoring Moving Range Query



**Publications and Submissions:**

1. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), Efficient monitoring of moving mobile device range queries using dynamic safe regions. In *The 11th International Conference on Advances in Mobile Computing and Multimedia*. MoMM'13.

2. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2014), Monitoring moving queries inside a safe region. In *The Scientific World Journal*. Hindawi.

# 5

# Monitoring Moving Range Query

With the emergence of mobile computing, monitoring continuous spatial queries over moving objects has become a requirement for many different daily applications, such as GPS, cargo tracking, traffic monitoring, location based networks and traveller information systems. However, wireless communication costs for location updates and query evaluation costs at the server end, represent the most common costs that determine the monitoring system performance.

The safe region technique is one of many proposed techniques that have been used to minimise the communication and computation costs of monitoring the moving range query. In the previous chapter (Chapter 4), we explained the safe region as an area where the set of objects of interest to the query does not change as long as the query remains inside it. Thus, there is no need to update the result of the query while it is roaming inside its safe region. However, when the query leaves its safe region the mobile device has to reprocess the query, thus communication with the server is needed and the query re-evaluation must re-commence. But, knowing when and where the mobile device will leave its safe region is widely known as a difficult challenge. In this chapter, we address this challenge and propose a novel method to monitor the position of the query

over time using a linear function based on the direction of the query obtained by periodic monitoring of its position.

The rest of the chapter is organised as follows. Section 5.1 gives our research motivation. Sections 5.2 and 5.3 describe our proposed method. Section 5.4 presents the algorithms used. Performance experiments are covered in Section 5.5. Section 5.6 summaries the chapter.

## 5.1 Motivation

The moving range query requires constant reporting of its result from the registration of the query to its cancellation. This is called the effective period of the query. Over this time, the query results must be continuously updated even if the query conditions remain unaltered during the effective period (Shengsheng & Chen, 2011). To reduce the updating costs while the query moves continuously, the safe region concept has been proposed (AL-Khalidi, Taniar, Betts, & Alamri, 2013; Cheema et al., 2011; Cho et al., 2013; Mokbel et al., 2004), which allows the query to report its current location and to request a new result only when it exits its current safe region. These two factors have the potential to significantly reduce communication and computational overheads. Because the query does not communicate with the server once it enters its safe region, neither the query nor the server will be aware of when and from which direction the query will leave its assigned safe region. Consequently, the server would not be able to calculate a new safe region for the query to roam in or update the result list without delay.

This chapter presents our technique for continuously monitoring a moving range query inside the Extended Safe Region. A linear function is proposed to monitor the moving query within its safe region. Since the Extended Safe

Region has an irregular shape, it is necessary to find a method to monitor the query inside it. Our method reduces the costs associated with communications in client-server architectures because an update of the location will be reported only when the query leaves its assigned safe region or upon the server's request. Computational results show that our method is successful in handling moving query patterns. We also present an analysis of the computation and communication costs of our algorithm which shows the advantages of our method. Our research presented in this chapter was published in (Al-Khalidi et al., 2014, 2013a).

## 5.2 Linear Motion Function to Monitor a Query inside Safe Region

This section presents our technique for continuously monitoring the moving range query inside the Extended Safe Region. A linear function of time model is proposed to monitor the moving query within its Extended Safe Region. In this technique, we will model the query positions as functions of time. In many circumstances, the area of the safe region is not of greatest significance, however, the essential point is when the query enters and leaves the safe region. At these events the server will be aware of the moving query location, allowing location updates to occur only when the query passes beyond its assigned safe region.

The safe region of the query is computed at the server side based on the intersections of the range objects. The computed safe region will be sent to the moving query from the server. Hence, the query will then send its location to the server when it moves outside its safe region. The query is aware of its

current location and its velocity, and in this case the query can calculate its next location(s).

$\bar{x}(t) = (\bar{x}_1(t),\ \bar{x}_2(t),\ ...,\ \bar{x}_d(t))$ represents the query's position at time $t$, assuming that the time $t$ is not before the current time. In order to model this position, we have used a linear function, which is specified by two parameters. The first parameter represents the position for the query at some specified time $t_{old}$, $\bar{x}(t_{old})$, termed the old position. The second parameter represents the velocity vector for the query, $\bar{v} = (v_1\ ,v_2\ ,\ ...,\ v_3)$. Thus, $\bar{x}(t) = \bar{x}(t_{old}) + \bar{v}(t - t_{old})$. The new position of the moving query is:

$$\bar{x}_{new} = \bar{x}_{old} + \bar{v}(t - t_{old}) \tag{5.1}$$

According to Equation 5.1, the query knows its position and, hence, there is no need to inform the server about its new location while moving within the safe region. Also, there is no need for the query to inform the server about its velocity when changing its speed and direction.

Generally, the query or the object positions are modelled as functions of time in order to make tentative near-future predictions to alleviate the problem of the frequent updates which will be required. Several studies (Jang et al., 2007; Jeung et al., 2010) have used this function to predict the path that the user will use; users may report these parameter values when their actual position deviates from what was previously reported according to some threshold. The prediction of the movement of the query's position can be made from the present into the far future. However, long-term prediction is not possible and short-term prediction suffers from the fork dilemma. Also, it is not usual for a query to exist for a long period of time within a useful threshold of its predicted movement. Therefore, if this query does not report its new position

and velocity, after some time, its old positional information will be inaccurate and not useful. Hence, this information will expire.

In our technique, the users do not need to report their parameter values when their actual position deviates from what they have previously reported. Users need to report their new position only when they leave the safe region. After reporting their new position, the server should calculate the new safe region depending on the current user's location and send it to the user. To avoid downlink in the communication and to insure that the server has received the new user's location, we allocate a time stamp parameter, $t_{stp}$, whereby the user should receive a new safe region within $t_{stp}$. Our technique does not need any prediction and it overcomes the problem of the fork dilemma. Monitoring of a moving query inside an Extended Safe Region has four scenarios:

1. The query is inside a safe region of one object (the shape of the safe region will be a circle).

2. The query is inside a safe region formed by two overlapping objects, both objects within the result list, (the shape of the safe region has two convex edges).

3. The query is inside a safe region formed by more than two overlapping objects ($n$ objects, $n > 2$), but all the objects are within the result list (the shape of the safe region has $n$ convex edges).

4. The query is inside a safe region formed by more than one overlapping object ($m$ objects, $m > 1$), but some of the objects are within the result list ($n$ objects, $n \geq 1$), and the rest are not within the result list ($r$ objects, $r = m - n \geq 1$) (the shape of the safe region has $n$ convex edge(s) and $r$ concave edge(s)).

### 5.2.1 Query within One Object

In some cases, the Extended Safe Region has a circular shape with a range of one object *pi* only. The location of the object *pi* will represent the centre of the Extended Safe Region, and $e$ will represent the radius. Note that the radius $e$ is the range of the query $q$. In Figure 5.1, the safe region of the query $q$ represents the whole range of the object $p3$, where $e$, $MinDist(p3,q)$ and the velocity of $q$ are known in advance. Figure 5.1(a) shows that $q$ can move from its current location in any direction $(d1,d2,...,dn)$ depending on its velocity; we will use the triangle solving method to calculate when $q$ will leave its safe region.

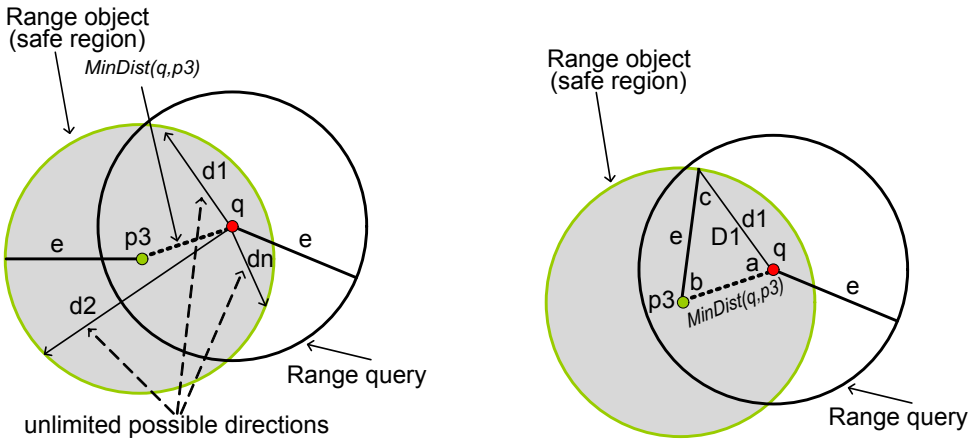
In Figure 5.1(b), we consider that $q$ will follow the direction $d1$. Using the triangle solving method, we calculate $D1$ (i.e., $D1$ is the distance between the query and the border when the query follows the direction of $d1$) using the triangle shown in this figure. This triangle has the side lengths ($D1$, $e$ and $MinDist(p3,q)$) and the internal angles ($a$, $b$ and $c$). To find $D1$ (the distance when the $q$ will be outside its safe region) the angle, $a$, between two sides, $MinDist(p3,q)$ and $D1$ should be found first.

The formulas for solving the triangles are:

$$c = arcsin(MinDist(pi,q) \times \frac{sin(a)}{e}) \tag{5.2}$$

The sum of the internal angles of the triangle is $180^\circ$

$$a+b+c = 180^\circ \longrightarrow b = 180-a-c \tag{5.3}$$




(a) Unlimited directions (b) Solving the triangle method

Figure 5.1: Safe region using linear function

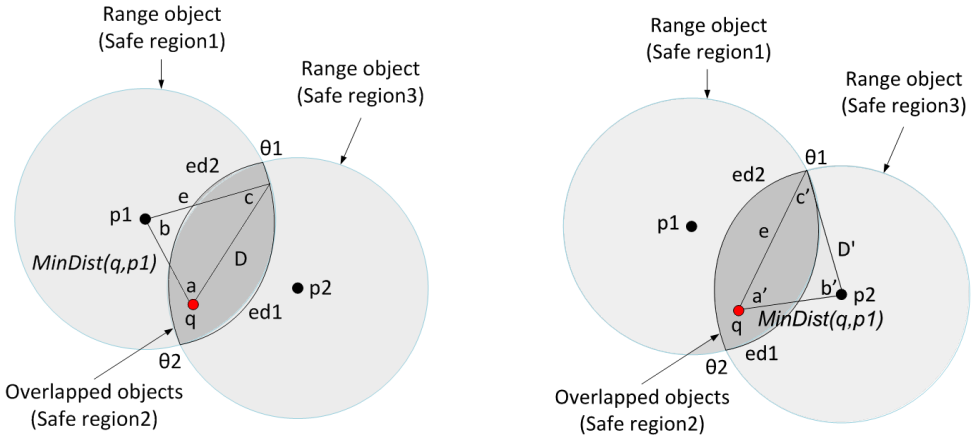
Now we can find $D$ by:

$$D1 = e \times \frac{sin(b)}{sin(a)} \tag{5.4}$$

Because the shape of the Basic and the Enhanced Safe Region are circular, then this monitoring method can be used to monitor the query in the Basic Safe Region either from the beginning when the query is in the centre of the safe region or when the query changes its direction and it is no longer inside the centre of the safe region. This monitoring method can also be used to monitor the query in the Enhanced Safe Region.

### 5.2.2 Query within Two Objects

In the next scenario, the Extended Safe Region might be created by two objects, both of which are within the result list (i.e., both of these two objects are within the range of the query). The overlapping for the ranges of these two objects will create an area which has two edges; both edges (curves) are convex. Figure 5.2, shows an example of two overlapping objects $(p1, p2)$ within the range of




(a) Crossing board $p1$ (b) Crossing board $p2$

Figure 5.2: Monitoring $q$ within range of two objects

the query. Each edge represents the border for the range of one object. The edge within distance $e$ from one object $pi$ ($pi \in \{p1, p2\}$) will be the border of that object.

The two convex edges ($ed1$ and $ed2$), which determine the Extended Safe Region are part of the range of the objects ($p1$ and $p2$). The query $q$ will be outside the range of an object ($pi$) when it crosses the edge of that object ($edi$). Crossing any edge of the Extended Safe Region by the query means that the query is leaving its current safe region and entering a new one. If we know in advance the direction of the query, then we can find which edge of the Extended Safe Region the query will cross. To monitor any query in this scenario, the intersection points between the range of the two objects ($p1$ and $p2$) should be found first (see Figure 5.2). Each curve (edge $ed1$, $ed2$) has a start and end angle. For example, the curve $ed1$ has ($\theta_1$&$\theta_2$) as start and end angles respectively, while the curve $ed2$ has ($\theta_2$&$\theta_1$) as start and end angle respectively.

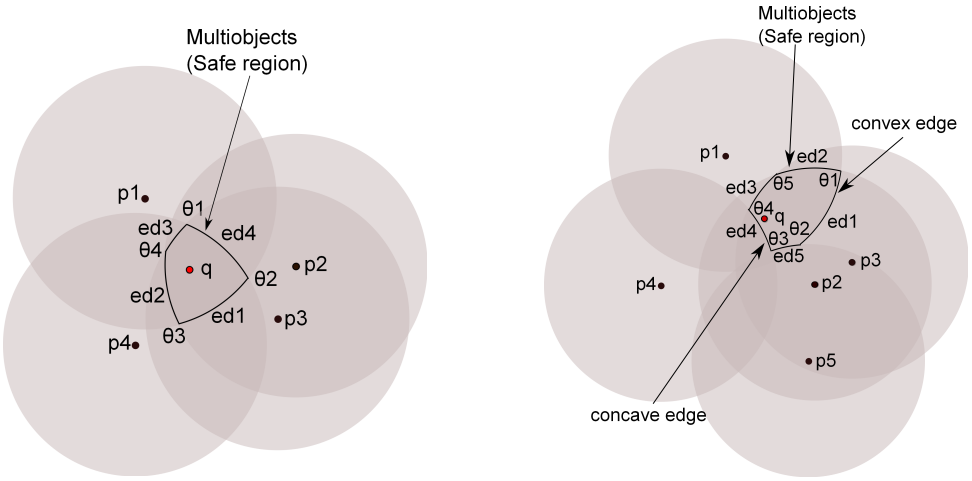
If the query $q$ is moving in a direction within the range of the angle ($\theta_1$ to $\theta_2$), then it will cross the edge of the object $p1$ ($ed1$) (see Figure 5.2(a)). If

the query $q$ is moving in the opposite direction, which is the direction within the range of angle ($\theta_2$ to $\theta_1$), then $q$ will cross the edge of the object $p2$ ($ed2$); see Figure 5.2(b). Equation 5.2 can be used to calculate the angle $c(c')$. After determining $c(c')$, the angle $b(b')$ can be found using Equation 5.3 (i.e., the summation of the internal angles of the triangle, which is 180°). The distance $D(D')$ that $q$ can move until it reaches the border of the Extended Safe Region can be found using Equation 5.4.

### 5.2.3 Query within Multi Objects

Monitoring the query in this scenario is very similar to monitoring the query which was presented in Section 5.2.2. However, we will add the start angle ($\theta_{si}$) and the end angle ($\theta_{ei}$) to each object ($pi$) surrounding the query. These angles are determined by the intersection of any object with any other one to specify the Extended Safe Region. The start and the end, ($\theta_{si}$) and ($\theta_{ei}$), angles respectively identify the corresponding edge of the range object bounding the Extended Safe Region.

The Extended Safe Region will be crossed from a specific edge. It is possible to know this edge based on the velocity of the query (speed and direction) and the starting and ending angles of the object $pi$. Figure 5.3(a) shows which edge is crossed by the query when it moves in a specific direction. Table 5.1 shows the query will cross $p1$ if it moves at an angle between (110° - 200°), and will cross $p2$ if it moves at an angle between (200° - 300°), to the rest of the table. Because interior angles must always add up to 360°, Equation 5.2 can be used to find out the angle ($c$), depending on the starting and ending angles of the corresponding edge.




(a) Safe region has only convex edges

(b) Safe region has convex and concave edges

Figure 5.3: Safe regions with different edges

Table 5.1: Start and end angles

| **Object** | $MinDist(q, pi)$ | $\theta_{si}$ | $\theta_{ei}$ |
|---|---|---|---|
| $p1\ (ed1)$ | 1.1 | 110° | 200° |
| $p2\ (ed2)$ | 1.3 | 200° | 300° |
| $p3\ (ed3)$ | 1.4 | 300° | 350° |
| $p4\ (ed4)$ | 1.2 | 350° | 110° |

### 5.2.4 Query within Multi Objects in/out the Result List

In some scenarios, the Extended Safe Region might be created from objects that are inside the result list or outside the result list. Parts of the border for the objects that are inside the result list will be the convex edges of the Extended Safe Region and those objects outside the result list will give the concave edges. Figure 5.3(b) shows the convex edge created by the borders of $p1, p2, p3$, and the concave edge created by the borders $p4, p5$. Monitoring the query in this case will occur through the following:

1. if the query is moving towards the convex edge, then Equation 5.2 is used to find the angle $c$;

2. otherwise, the query will move toward the concave edge and then Equation 5.5 is used.

$$c = 180 - arcsin(MinDist(pi, q) \times \frac{sin(a)}{e}) \tag{5.5}$$

The inverse sine function usually returns angles less than 90°. For example, Equation 5.2 uses the smallest angle because the third side of the triangle will always be inside the circle (safe region). However, when the third side of the triangle lies outside the safe region the larger angle must be used, see Equation 5.5. The distance between the query and the object *pi* is used to decide which of Equation 5.2 or 5.5 is used. Equation 5.2 is used when the objects within the range search are within distance $e$ from the query, while Equation 5.5 is used when the objects outside the range of the query are not within the distance $e$ from the query.

## 5.3 Support Arbitrary Moving Query

In our moving range query technique, the query has the ability to evaluate its location at all the times. The query will calculate its current location wherever the velocity of the query changes. By knowing its current location, the query will compute the distance or the time (i.e., when it will leave its safe region and from which edge). The problem of fork dilemma does not occur when the query is inside the safe region because a quick recalculation of the new direction can be made by the query itself and the set of objects of interest does not change.

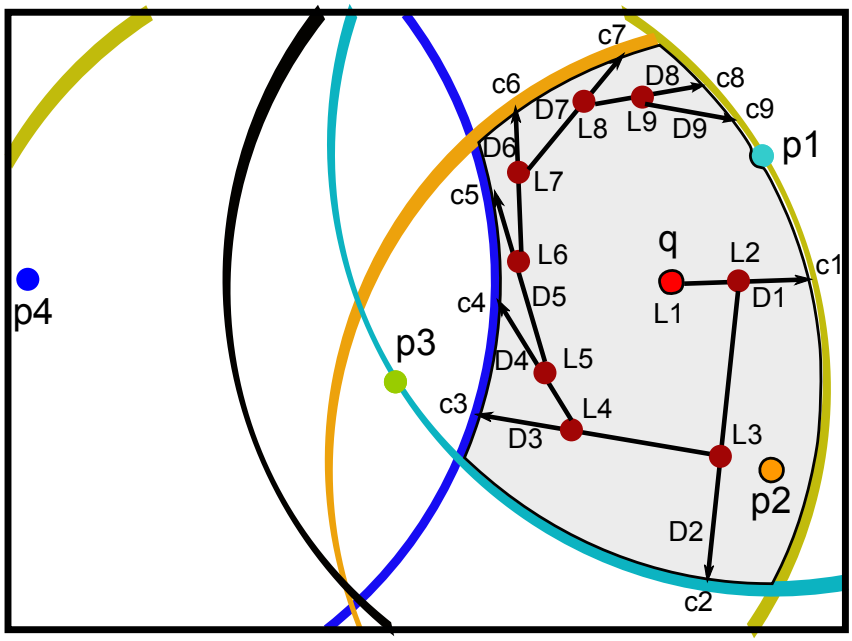



Figure 5.4: Arbitrary moving query inside safe region

Figure 5.4 shows that at each turn the query will calculate its distance to the boundary. The shaded area represents the safe region of the query $q$ and the location $L1$ represents its current location. Whenever the query changes its velocity, the query then will compute its new distance to the edge which will be crossed later. This figure shows that the query changes its velocity eight times in order to leave its safe region at the locations ($L2$, $L3$, $L4$, $L5$, $L6$, $L7$, $L8$, $L9$). In each location a new distance from the edge and a new point on the edge (which represents the place that the query will cross the edge) will be calculated by the query. For example, as shown in Figure 5.4, ($L1$) represents the current location of the query $q$ and ($D1$) represents the distance that the query needs to cross the edge of the safe region boundary in the point ($c1$) if the query does not change its direction. ($L2$, $D2$, $c2$) represents the new location of the query, the new distance and the new point on the boundary of the safe region respectively and so on with ($L3$, etc.).

## 5.4 Algorithm to Monitor Moving Query inside Extended Safe Region

An algorithm to monitor a moving query inside an Extended Safe Region is now presented (Algorithm 5.1). The safe-object list from Algorithm 4.3 is used to determine the objects that form the boundary of the safe region. Each object having a shared border with the Extended Safe Region will be registered in a table with the start and end angle of that border. To fill the table with the start and end angles, the safe-object list is constructed first. The list commences with the start and the end angle of the closest border (edge) to the query. The next edge, starting from the end of the previous edge is added to the list. This is continued until the boundary arc terminating at $360^{\circ}$ has been added to the list. If the query changes its direction at anytime, then its location will be considered. The new location of the query will be used to find out the distance that the query needs in order to leave its safe region and the point on the boundary at which the query will cross the safe region.

## 5.5 Experimental Results

Several experiments were conducted to evaluate our proposed algorithms. A synthetic dataset was used to test the proposed linear function. Three different density environments were created (low = 100 objects, medium = 300 objects and high = 500 objects) to measure the performance of our monitoring method in a data space of $100km \times 100km$.

1000 queries were randomly generated in low, medium, and high density environments. The average distance the query moves until leaving its safe region was then recorded. The average distance from the current location of

**Algorithm 5.1.** Monitoring moving range query algorithm

1: /* $q$: query point, $v$: the velocity of $q$ and $e$: the Euclidean distance threshold */
2: /* Find when $q$ will leave its safe region */
3: LIST $SOL$
4: Call Extended Safe Region algorithm to find safe-object list ($SOL$)
5: From $SOL$, find the closest border ($Cedx$) of $px$ to $q$
6: Find the start and end angles of $Cedx$ and add them to the table of start and end angles
7: $TCed = Cedx$
8: **for** each $pi$ in $SOL$ **do**
9: **if** $TCed$ intersects with the border of $pi$ **then**
10: $pi$ should share an angle with $TCed$
11: add $edi$ and its angles to start and end angle table
12: $TCed = edi$
13: **else**
14: Ignore this $pi$
15: **end if**
16: Exit for, if the table completed 360° degree
17: **end for**
18: Monitoring $q$ inside safe region
19: **for** each $edi$ in the start and end angle table **do**
20: **if** $MinDist(edi,q) < e$ **then**
21: the curve is convex
22: $c = arcsin(MinDist(pi,q) \times \frac{sin(a)}{e})$
23: **else**
24: the curve is concave
25: $c = 180 - arcsin(MinDist(p3,q) \times \frac{sin(a)}{e})$
26: **end if**
27: $b = 180 - a - c$
28: $D = e \times \frac{sin(b)}{sin(a)}$
29: **end for**
30: **if** $q$ changes its direction at anytime before it crosses any $edi$ in the start and end angle table **then**
31: goto 17 and use the new position of $q$
32: **end if**

the query to the border of the current safe region was also recorded in each experiment.

Figure 5.5 is a screenshot of the software used to calculate the distance that a moving range query can travel before it leaves its safe region when it

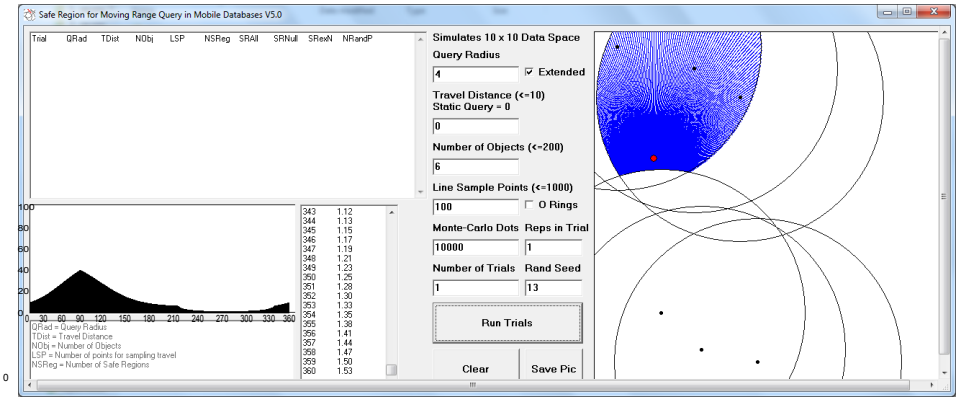



Figure 5.5: Distance that a query travels before leaving its safe region

moves in any direction. This figure shows the distance between the query and the safe region border in all directions. The source of our implementations can be downloaded from the following URL:`https://www.dropbox.com/sh/b0vuicc0t3ymhq4/AADxzv9UeuLsyzMrlfxSMj3ra`.

### 5.5.1 Moving Query in Different Environments

Figure 5.6 shows a comparison of three different objects' environments: low, medium and high density, with a radius of range search that varies from 5$km$ to 30$km$.

We found that the average distance the query could roam until crossing the border of the extended safe region is high when there are a few objects surrounding it. The reason is that, in most cases, there is no object within the range of the query and there is a long distance until one object becomes within the range of the query. This gives the query a very good indication of the distance it can travel before the query's set of objects of interest will change, or the time until the query will find the first new object when its current list of objects of interest is empty. In the range query, when a user invokes a query within a specific range and the result returned is null, the user needs to enlarge

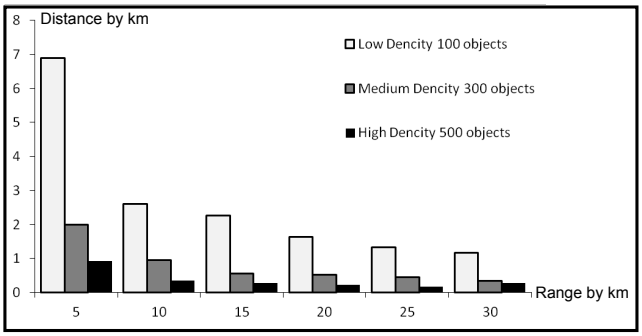



Figure 5.6: The average distance the query can move until its objects of interest change in different density environments

the size of the range. However, in our method the user will have an indication about how far away the nearest object to the query is, and in what direction from the query.

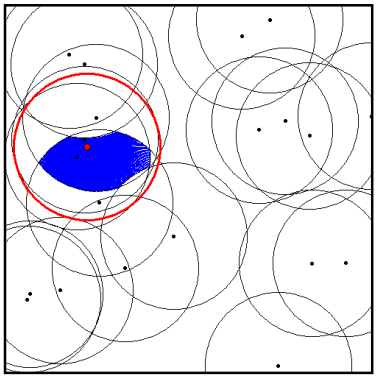
### 5.5.2 Case Studies

In this section we present two case studies to explain our method for: *i*) finding when the set of objects of interest will change (when there is already object(s) in the set), and *ii*) finding the nearest object in any direction (when there is no object in the result list).

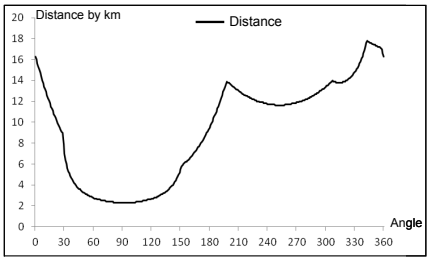
#### Objects Surrounding the Query

This case study considers a query which has objects of interest in its range. Our linear motion function can inform the query about the distance that the query needs to pass in any direction before its set of objects of interest will change.

Figure 5.7 shows a case study of a query with objects surrounding it. In this case the range query is $20km$ and the number of objects in the dataset is 20. Figure 5.7(a) shows that there are six objects surrounding the query in that specific moment (four objects in the set of objects of interest and two just outside the range). Figure 5.7(b) shows the shortest distance ($2.3km$) the query



(a) Different directions the query may follow




(b) The distance from the query to the border of the safe region by any direction

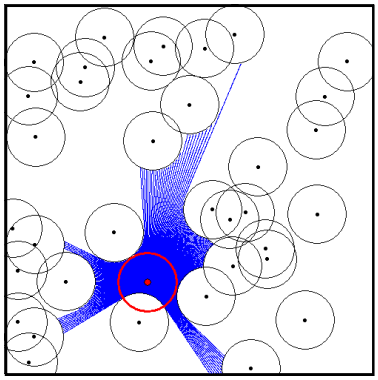
Figure 5.7: Distance that query can travel before the objects of interest change

can move until the set of objects of interest will change and that occurs when the query moves within an angle of (81°-103°). The longest distance the query can move until the set of objects of interest change is (17.8$km$) at an angle of (343°).

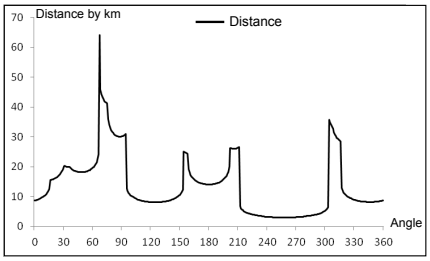
### No Object within the Range Query

This case study considers a query which has no object of interest in its range. Our linear motion function can inform the query about the distance that the query needs to pass in any direction before finding the first object.

Figure 5.8 shows the distance in any direction to the nearest objects from the query. In this case study the range query is 8$km$ and the number of objects in the dataset is 35. Figure 5.8(a) shows that there is no object within the range of the query at that specific moment. Figure 5.8(b) shows that the shortest distance the query can travel to find an object of interest is in the direction (246°-269°) in which case the object will be within 3.1$km$ from the query. While the longest distance to find an object of interest will be in the direction of the angle (67°) in which case the object will be 64.1$km$ away from the query.



(a) Different directions the query may follow




(b) The distance from the query to the first object by any direction

Figure 5.8: Distance of the closest object in all directions from the query

## 5.6 Summary

In this chapter, we propose a linear model to monitor the moving range query inside a safe region for mobile navigation. This approach predicts when the query will leave the safe region based on its current location and velocity. The aim of this technique is to *i*) reduce the need for continuous monitoring of the query, and *ii*) to eliminate the need for the user to follow a defined path. The method is used by the query whenever the server allocates it to a new safe region. Our method does not suffer from the problem of fork dilemma because it is not calculated as a linear function of time alone. By contrast, in our method we use the time and the concept of the safe region together, hence, if the query makes a sudden turn, the result will not be affected because the query will still be located inside the safe region.

# Chapter 6

# Lookforward Moving Range Query



**Publications and Submissions:**

1. AL-Khalidi, H. Taniar, D. Nguyen, K. Betts, J. and Alamri, S. (2014), Lookforward moving range search query based on road network. In *Computers & Mathematics with Applications*. Elsevier. (Under Review)

2. Alamri, S. Taniar, D. AL-Khalidi, H. and Nguyen, K. (2014), Density-based for moving query in multi-floor indoor spaces. *IEEE Transactions on Knowledge and Data Engineering*. IEEE. (Under Review).

# 6

# Lookforward Moving Range Query

Moving range query is one of the most common queries in spatial databases, where a user invokes a query to find all the surrounding objects of interest while s/he is moving. Most studies of moving queries consider Euclidean distances to retrieve the results with low cost, but with poor accuracy (i.e., Euclidean distance is less than or equal to network distance). Thus, researchers show that a moving query using network distance retrieves the results with high accuracy but with a vast amount of network distance computations and very high amount of communication between the query and the database server. However, both of these techniques retrieve all objects in a given radius from the query centre. In many situations, however, retrieving all objects is not necessary, especially when some of these objects are not important to the users or because they may force the users to deviate from their original route. As well, objects that appear in the result list for a very short time should be eliminated because they do not give the user enough time to react and make an appropriate decision. In response, we propose a new query technique based on the spatial road network called lookforward moving range query, which makes the user concentrate only on objects of interest in the forward space of the query point.

The rest of the chapter is structured as follows. Section 6.1 illustrates our motivation. Section 6.2 presents some preliminary concepts used in this paper. This will be followed by Section 6.3, which will describe our proposed technique, the lookforward moving range query. Section 6.4 explains our method of simplifying the path query. The performance evaluation of the approaches is given in Section 6.5. Section 6.6 concludes the chapter.

## 6.1 Motivation

In recent years, mobile technology has been advancing quickly and the demand for mobile information services and mobile queries is growing. Many mobile queries have been proposed and studied over the years; most of these studies consider Euclidean spaces, where the distance between two objects is determined by their relative position in space. However, in practice, the trajectory between two objects is specified by the underlying network (such as roads and railways). Thus, measuring the actual network distance between two objects (the length of the shortest road connection between them) is more important than measuring their Euclidean distance. Spatial network databases research, (Nguyen & Cao, 2012; Papadias et al., 2003; Safar, 2005; Taniar et al., 2011; Tran et al., 2009), has focussed on developing efficient algorithms that expand the spatial query processing methods by integrating connectivity and location information.

In this chapter, we focus on a special type of moving range query based on road network. Our work is motivated by the following fact: all traditional range queries try to find all objects of interest around the query point without controlling the query direction. For example, the user may invoke a query to find all "shopping centres" within $4km$ along his moving path from $S$ to $D$, ignoring any shopping centre that is left behind. In such a request, the query

path direction and the user's location are two crucial conditions that need to be considered. Therefore, we propose a novel query processing technique for the moving range query in spatial network databases, called lookforward moving range query (*LMR*).

In this technique, we introduce a new way to distinguish between the significant important objects and the minor important objects inside the boundary of the moving range query. Only objects that are located in the direction of the user while he/she is moving are recorded, excluding that are behind the moving user or will divert the user away from his/her trajectory. We also improve the selectivity of the filter step to reduce the number of candidate objects, and consequently, minimise the number of communications between the mobile device and the database server. The resulting technique achieves a better running time and delivers a better performance, yet with low split points. To the best of our knowledge, this is the first work that excludes of some objects inside the boundary of the range query, which are unimportant to the user, to achieve processing effectively.

## 6.2 Approximate Trajectory Techniques - Preliminaries

Our approach is based on moving range query processing in spatial road networks. We have employed the advantages of an approximate trajectory to achieve a better result for forward objects of interest.

The moving query follows a path through space as a function of time. This path is called a trajectory. To capture the accurate and complete trajectory of a moving query a massive amount of data is needed. Therefore, there is a need

to reduce the size of the data required to store a trajectory in order to save storage costs and reduce excessive data.

In previous research, trajectory data reduction techniques have been used to reduce the data size of trajectory representation without greatly reducing accuracy. With such techniques, if the inaccuracy of the user's location is beyond the application conditional threshold (error tolerance), then the location of the user needs to be reported to the location server (database server).

There is a specific rate of allowable inaccuracy that is used to deal with the positions of points in multidimensional space. Distance-based error measure has been used to specify the allowable inaccuracy rate (Alavi & Pahlavan, 2003).

If we consider the error threshold used in online data reduction techniques, we will find that measuring the error threshold can be done using the aggregation distance between the approximated trajectory and the original trajectory. Hence, the distance between a location on the original trajectory gained from the positioning mechanism is very close to an estimated location on the approximated trajectory.

Therefore, measuring the error can be done by computing the average or the total perpendicular Euclidean distances from each of the sampled location points in the original trajectory to the approximated trajectory. Perpendicular Euclidean distance is used in the Douglas Peucker algorithm as the error measure (Douglas & Peucker, 1973; Hershberger & Snoeyink, 1992). This algorithm is one of the main methods used to simplify the original trajectory (approximate trajectory), such that the simplified trajectory has fewer points and deviates from the original trajectory by at most $\gamma$ (the error threshold) at any point.

There are two types of approximate trajectory algorithms, batch and online. Batch algorithms require the availability of a full data series, while the online algorithms do not. Online algorithms are used to approximate data streams in

real time. Batch algorithms always produce better quality results when compared to online algorithms. Approximation algorithms can be grouped into one of the following three categories (Meratnia & By, 2004); top-down, bottom-up and sliding window. Top-down and bottom-up algorithms are classified as batch algorithms while sliding window algorithms are classified as online algorithms. We will review these algorithms below.

### 6.2.1 Top-Down Approximate Technique

In this algorithm, the main idea is to replace the original trajectory by an approximate line segment. If the replacement does not meet the error requirement, then a recursive process will partition the original problem into two sub-problems by selecting the location point contributing the most errors as the division point. This process will continue until the error between the approximate trajectory and the original trajectory is below the specified error threshold.

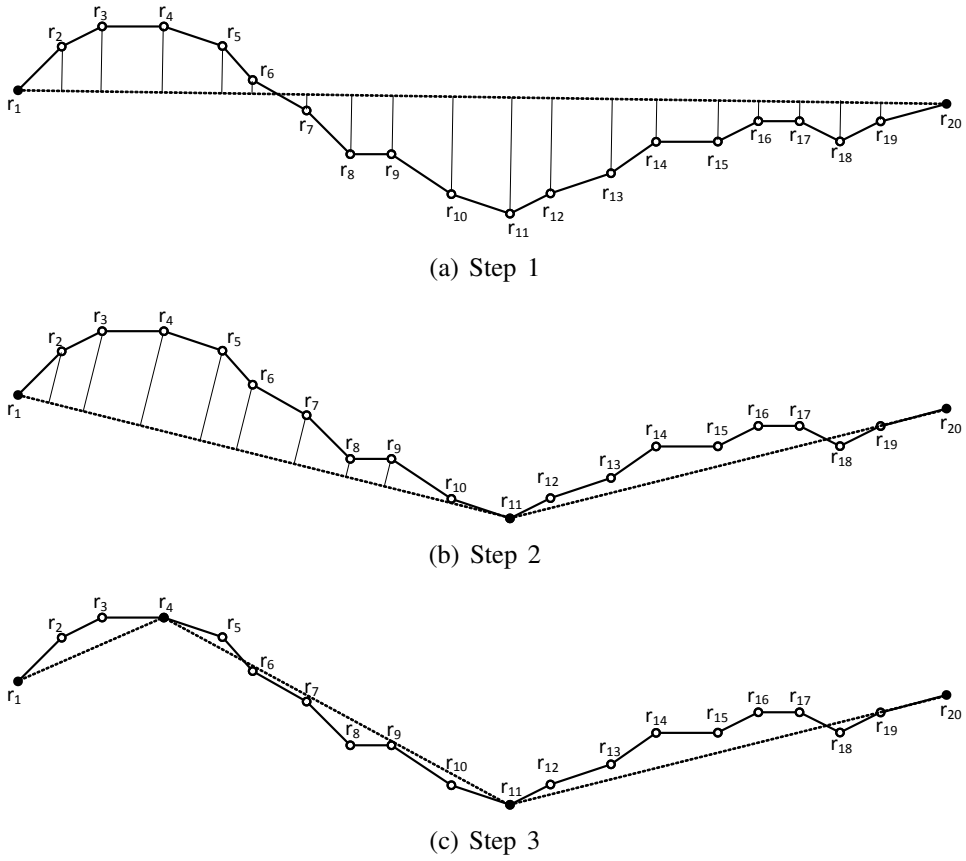
The Douglas Peucker (DP) algorithm is one of the most popular top-down methods and is often used, and was originally proposed for, line simplification. This algorithm tries to keep the directional trends in the approximation line using a distance threshold (error threshold), which may be changed according to the required simplification amount.

The original trajectory can be represented as a polyline $L = \{r_1, r_2, ..., r_n\}$ and the error tolerance can be represented as $\gamma$, where $r_i$ is the location of the moving query at time $t_i$. The Douglas Peucker algorithm is used to obtain a simplified trajectory. Initially, the algorithm selects the first point and the last point of the data series of the trajectory as the anchor point ($r_a$) and the float point ($r_f$) respectively, and constructs a straight line between the anchor point ($r_a$) and the float point ($r_f$) $\overline{r_a r_f}$. For all data points between the anchor and the

float, the perpendicular distance to the constructed line connecting anchor and float points is defined, then the algorithm finds the point $r_i \in L$ farthest from the line. If the Euclidean distance is $D_E(r_i, \overline{r_a r_f}) \leq \gamma$, then the segment $\overline{r_a r_f}$ is reported as the simplified trajectory. Otherwise, the line ($\overline{r_a r_f}$) is divided at the data point that causes the farther distance ($r_i$). This divided point ($r_i$) becomes the new anchor point for the second segment, and the float point for the first segment. The algorithm recursively examines the new sub-trajectories $\{r_a, ..., r_i\}$ and $\{r_i, ..., r_f\}$, reporting the concatenation of their simplified trajectories as the simplified trajectory (Zheng & Zhou, 2011).

The Douglas Peucker algorithm is illustrated in Figure 6.1 to find the approximate trajectory from $r_1$ to $r_{20}$. As shown, in step 1 (see Figure 6.1(a)), the starting point $r_1$ and end point $r_{20}$ are selected as the anchor point and the float point respectively, to generate an approximate line segment $\overline{r_1 r_{20}}$. We derive the perpendicular Euclidean distance from each sampled location point on the original trajectory to the approximate line segment $\overline{r_1 r_{20}}$. The sampled location point deviating the most from $\overline{r_1 r_{20}}$, i.e. $r_{11}$ in this example, is chosen as the division point because some of the perpendicular error distances are greater than the error tolerance. In the second step of the algorithm (see Figure 6.1(b)), the trajectories $\overline{r_1 r_{11}}$ and $\overline{r_{11} r_{20}}$ are used to approximate the original trajectory. In this step, the original problem is divided into two sub-problems where the line segment $\overline{r_1 r_{11}}$ is to approximate the sub-trajectory $\{r_1, r_2, ..., r_{11}\}$ and the line segment $\overline{r_{11} r_{20}}$ is to approximate the other sub-trajectory $\{r_{11}, r_{12}, ..., r_{20}\}$.

As shown, in the first sub-problem, several sampled location points have their perpendicular error distances to $\overline{r_1 r_{11}}$ greater than the error tolerance. Therefore, $r_4$, the sampled location point deviating the most from $\overline{r_1 r_{11}}$, is chosen as the division point. Then, the line segment $\overline{r_1 r_4}$ is to approximate the sub-trajectory $\{r_1, r_2, ..., r_4\}$ and the line segment $\overline{r_4 r_{11}}$ is to approximate the




(a) Step 1

(b) Step 2

(c) Step 3

Figure 6.1: Top-down algorithm using Douglas Peucker algorithm

other sub-trajectory $\{r_4, r_5, ..., r_{11}\}$. As a result, the division sub-trajectories are processed recursively until all the sampled location points have perpendicular distances to their approximate line segments within the error threshold.

Returning to the second sub-problem, the perpendicular distances of all the sample points to the line segment $\overline{r_{11}r_{20}}$ are smaller than the error tolerance. Therefore, further dividing is not required. Thus, the final approximated trajectory is $\{r_1, r_4, r_{11}, r_{20}\}$.

The Douglas Peucker algorithm, thus, uses the points that deviate the most to approximate the trajectory in order to reduce error. However, this algorithm cannot guarantee that the selected division points are the best choices (Zheng & Zhou, 2011).

### 6.2.2 Sliding Window Approximate Technique

The online approximate algorithms are not relevant and practical as batched algorithms. It is true that batched approximate algorithms produce excellent approximations due to the access of the whole trajectory, but this will not prevent the fact that online algorithms are functional in realistic applications. The sliding window algorithms are used for time series data mining, therefore, it can be applied in trajectory approximation. The basic concept of this algorithm is to fit the location points with a valid line segment in a growing sliding window. Then, the algorithm proceeds in growing the sliding window and its matching line segment until reaching an approximation error that exceeds certain error limits. The algorithm starts by initialising the first data point of the trajectory as the anchor point $r_a$ and the third data point in the trajectory as the float point $r_f$. As long as all perpendicular Euclidean distances of all data points in the window (i.e., between the anchor and the float points) are below the error threshold, the sliding window will continue to expand (i.e. by including the next data point).

When a new data point $r_i$ is added to the sliding window, the line segment $\overline{r_a r_i}$ is used to fit the sub-trajectory containing all the location points within the sliding window. The sliding window expands by including the next location point $r_{i+1}$, as long as the distance errors for all the location points contained in the sliding window, derived against the potential line segment $\overline{r_a r_i}$, are smaller than the error threshold $\gamma$. Otherwise, two strategies can be applied: either,

- Before Opening Window *BOPW*
- Normal Opening Window *NOPW*

which are introduced by Meratnia and de By (Meratnia & By, 2004).

In the BOPW strategy, the last valid line segment $\overline{r_a r_{i-1}}$ is included as part of the approximated trajectory, $r_{i-1}$, $r_{i+1}$ and $\{r_{i-1}, r_i, r_{i+1}\}$ are set as the new anchor point, the new float point and the new sliding window, respectively. The algorithm continues to visit all the location points in the original trajectory.

Figure 6.2 illustrates the Before Opening Window algorithm. First, $r_1$, $r_3$ and $\{r_1, r_3\}$ are set as the anchor point, the float point and the initial sliding window, respectively. The sliding window expands into $\{r_1, r_2, r_3, r_4\}$ since $\overline{r_1 r_3}$ fits $\{r_1, r_2, r_3\}$ very well. Once more, all the location points within the sliding window $\{r_1, r_2, r_3, r_4\}$ do not have a distance error greater than the pre-defined error threshold, i.e. $\overline{r_1 r_4}$ fits $\{r_1, r_2, r_3, r_4\}$ sufficiently well. Thus, the algorithm continues to expand the sliding window into $\{r_1, r_2, r_3, r_4, r_5\}$. Since $\overline{r_1 r_5}$ fits $\{r_1, r_2, r_3, r_4, r_5\}$ very well, the sliding window expands into $\{r_1, r_2, r_3, r_4, r_5, r_6\}$. At this time, the distance errors for some location points in the sliding window are greater than the error threshold, i.e. $r_3$ and $r_4$, ($\overline{r_1 r_6}$ does not fit $\{r_1, r_2, r_3, r_4, r_5, r_6\}$). As a result, the last valid line segment, i.e. $\overline{r_1 r_5}$, is granted as a part of the approximated trajectory. Afterward, the anchor point and the sliding window are reset as $r_5$ and $\{r_5, r_7\}$, respectively, by adding the next data point ($r_7$) into the sliding window to be the float point. The algorithm continues to process the rest of the trajectory and then eventually chooses $\{r_1, r_5, r_{12}, r_{20}\}$ as the approximated trajectory, where the sub-trajectories $\{r_1, r_2, r_3, r_4, r_5\}$ to fit with $\overline{r_1 r_5}$, $\{r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}\}$ to fit with $\overline{r_5 r_{12}}$ and $\{r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{18}, r_{19}, r_{20}\}$ to fit with $\overline{r_{12} r_{20}}$. Thus, the final approximated trajectory is $\{r_1, r_5, r_{12}, r_{20}\}$.

The NOPW strategy chooses location points within its sliding window, which have the highest distance error as the closing point of the approximating line segment and at the same time as the new anchor point. NOPW applies the Douglas Peucker algorithm on the data in the window. This trail and error

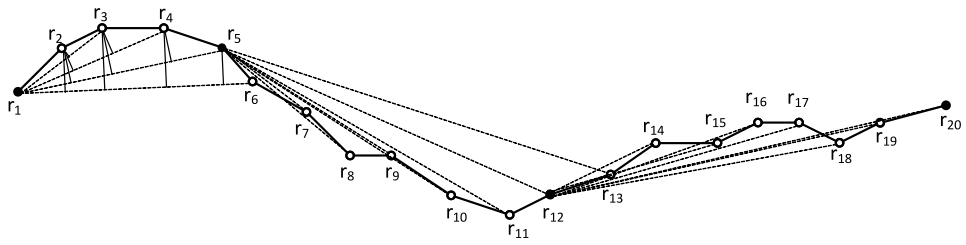



Figure 6.2: Sliding window algorithm using BOPW algorithm

works very well in reducing the approximation error. With the new anchor point $r_a$, the NOPW algorithm continues to process the rest of the trajectory.

Figure 6.3 illustrates the NOPW algorithm. First, $r_1$, $r_3$ and $\{r_1, r_2, r_3\}$ are set as the anchor point, the float point and the initial open window, respectively. Similar to the explanation in Figure 6.2, the segments $\overline{r_1 r_3}$, $\overline{r_1 r_4}$ and $\overline{r_1 r_5}$ respectively fit the windows $\{r_1, r_2, r_3\}$, $\{r_1, r_2, r_3, r_4\}$ and $\{r_1, r_2, r_3, r_4, r_5\}$ very well, because all the location points within these windows do not have distance error greater than the pre-defined error threshold. When the sliding window expands into $\{r_1, r_2, r_3, r_4, r_5, r_6\}$, the errors for $r_3$ and $r_4$ are greater than the error threshold. Instead of choosing $r_5$ as the closing point (i.e., BOPW algorithm does), the NOPW algorithm chooses $r_3$ as the closing point (i.e., similar to DP algorithm) to include the line segment $\overline{r_1 r3}$ as a part of the approximated trajectory ($r_3$ is deviating the most from $\overline{r_1 r_6}$). Then, the anchor point is reset as $r_3$ and the sliding window is reset as $\{r_3, r_4, r_5, r_6\}$. The algorithm continues to process the rest of the trajectory and then eventually chooses to fit $\{r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}\}$ with $\overline{r_5 r_{11}}$ and $\{r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{18}, r_{19}, r_{20}\}$ with $\overline{r_{11} r_{20}}$. Thus, the final approximated trajectory is $\{r_1, r_3, r_{11}, r_{20}\}$.

Top-down and sliding window algorithms are computationally expensive. The time complexity of both algorithms is $O(N^2)$ with $N$ being the number of data points (Meratnia & By, 2004).

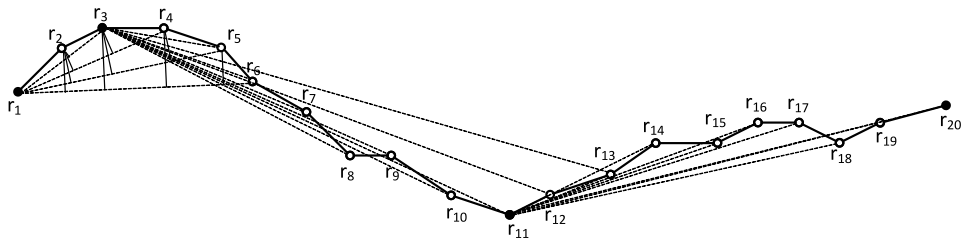



Figure 6.3: Sliding window algorithm using NOPW algorithm

## 6.3 Lookforward Moving Range Query

Lookforward moving range query is a conditional moving range query which restricts the results (objects of interest) in the forward direction of the query path. We design a new method, called *lookforward moving range query* LMR which is based on the spatial road network to eliminate some objects of interest within the range query which are less important to the moving user. This elimination will reduce the number of the objects of interest in the result list, leading to a reduction in the number of split points along the path of the query. This reduction will positively affect the number of updates for the query location.

The main problem with the moving range query based on the road network, is the multiple number of communications between the mobile device and the database server. Moreover, the users cannot make a decision due to the fast updating of the search result in areas with a high density of objects. Also, many users do not like to divert their route to heading back and away from there original path. To solve such problems, we consider a new type of moving query called lookforward moving range query (LMR) which is based on the spatial road network. In this technique, we eliminate all objects of interest that are less important to the user. The user will only focus on objects ahead of them while they are moving on their path. Any object that will make the user

divert back from their original path will be excluded, at the same time critical objects will also be excluded, to give the user sufficient time to make a decision.

The idea of LMR is to divide the entire query path into segments where each segment represents a link between two consecutive road intersections, then to implement the search at both ends of each segment to divide each segment into sub-segments. Each sub-segment is a link between either consecutive two split points or between a split point and an adjacent intersection.

Generally, our LMR technique has six steps to find the objects of interest: *filter step, refinement step, expansion step*, *half space step*, *split step* and *result step*. The half space step is particularly important for cleansing objects of interest and give the moving query a perfect result. Full details of our proposed approach LMR and its algorithm are presented below.

### 6.3.1 Filter Step and Refinement Step

The filter step and refinement step in the lookforward moving range query is similar to the filter step and refinement step in the static range query (see Section 2.1.1 Static Range Query). However, in the static range query, the query $q$ is a point and when the search starts traversing the R-tree, then a prune will occur to eliminate all MBRs with $MinDist$ to the query point $q$ that are greater than $e$, whereas, in our technique the query $q$ is a path ($[S,D]$) from a road network. MBRs with Euclidean minimum distance $MinDist$ to $q = [S,D]$ greater than or equal to $e$, will be pruned.

Figure 6.4 shows an example of the first two steps (filter and refinement) of our proposed LMR technique which clearly explains the filter step and refinement step. $SD$ is a path from the road network which represents the query path $q = [S,D]$; starting at $S$ and ending at $D$. If we assume the radius $e$ of the

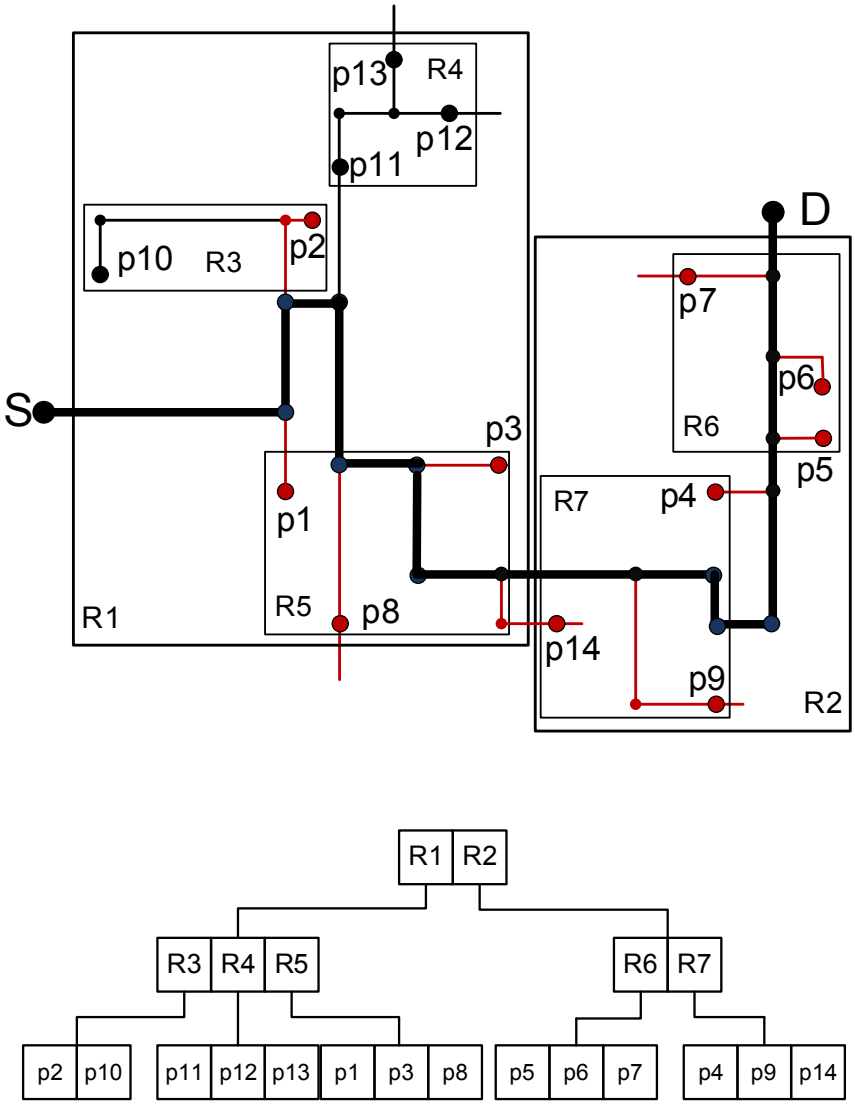



Figure 6.4: Filter step and refinement step

query is equal to 5, then in these steps (filter and refinement) all the MBRs, objects and segments, that fall farther away than the Euclidean distance $e(5)$ from the query path $(S, D)$ are pruned. In our technique we have added another prune: each item $(X)$ located Euclidean distance $e$ from the query path will also be pruned (e.g., the MBR $R4$ and the object $p10$), where $X$ is either an object or an $MBR$. As we know $D_N \geq D_E$, which means there is a high probability this object is located outside the network distance $e$. On the other hand, if

Table 6.1: Minimum distance between the object and the query path $[S,D]$

| **Object** | **Euclidean Distance** ($D_E$) | **Network Distance** ($D_N$) |
|---|---|---|
| $p1$ | 3 | 3 |
| $p2$ | 3 | 4 |
| $p3$ | 3 | 3 |
| $p4$ | 2 | 2 |
| $p5$ | 2 | 2 |
| $p6$ | 2 | 3 |
| $p7$ | 3 | 3 |
| $p8$ | 3.6 | 6 |
| $p9$ | 3 | 8 |
| $p10$ | 5 | 16 |
| $p11$ | 5 | 5 |
| $p12$ | 8 | 11 |
| $p13$ | 9.2 | 11 |
| $p14$ | 2 | 5 |

$D_N = D_E$ of this object then this object will be a critical object. A critical object is an object that enters the range search for a very short period and the distance between its two split points is zero or almost zero. The result will be $\{p1, p2, p3, p4, p5, p6, p7, p8, p9, p14\}$ and the segments that go through these objects. These objects will be in the filter list $FL$, $\forall pi \in FL, (D_E(pi,q) < e)$, and they should pass the next three steps to enter the result list and become objects of interest. Table 6.1 shows the Euclidean distance $D_E$ and network distance $D_N$ between each object; and the query path $[S,D]$ depends on Figure 6.4.

### 6.3.2 Expansion Step

The expansion step examines the query path $[S,D]$ intersection-by-intersection. A segment is an interrupted path from one intersection to another intersection. The example in Figure 6.5 has only objects and segments that have passed the previous steps. This figure is derived from Figure 6.4. Figure 6.5 shows

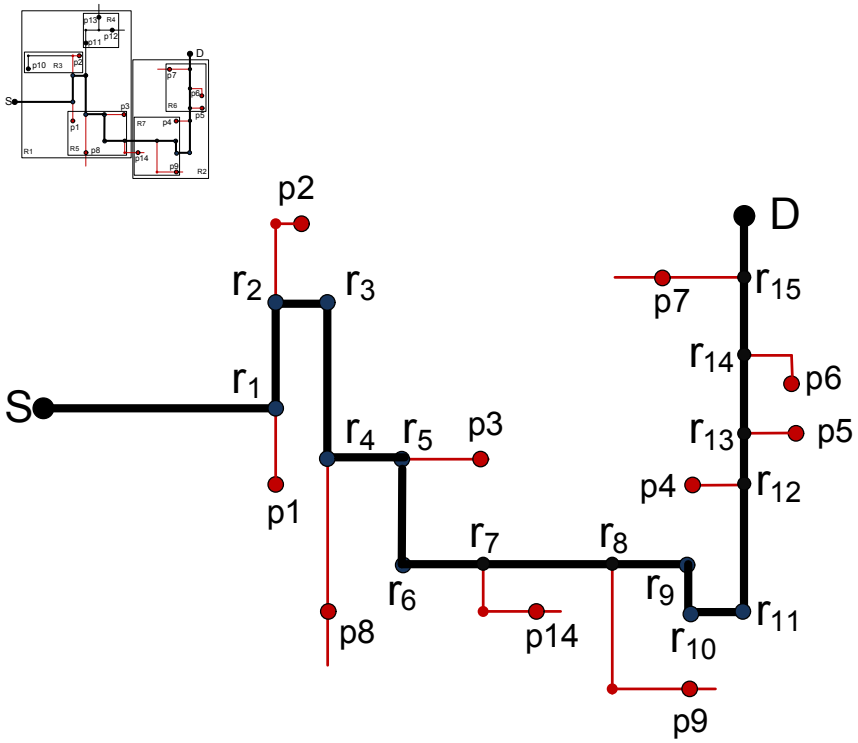



Figure 6.5: Expansion step

segments that start from point $S$ to intersection $A$, from intersection $A$ to intersection $B$, and also from intersection $B$ to intersection $C$, and so on until destination point $D$.

The expansion step, in our technique, goes through two steps to achieve the final result: *i)* segmenting the query path, and *ii)* processing the LMR query for each intersection.

### Segmenting the Query Path

Segmenting the query path means breaking the path $[S,D]$ into segments as follows: $(S,r_1,9)$, $(r_1,r_2,4)$, $(r_2,r_3,2)$, $(r_3,r_4,6)$, $(r_4,r_5,3)$, $(r_5,r_6,4)$, $(r_6,r_7,3)$, $(r_7,r_8,5)$, $(r_8,r_9,3)$, $(r_9,r_{10},2)$, $(r_{10},r_{11},2)$, $(r_{11},r_{12},5)$, $(r_{12},r_{13},2)$, $(r_{13},r_{14},3)$, $(r_{14},r_{15},3)$ and $(r_{14},D,2)$, where the separated number represents the length of the segment.

**Expansion Processing for Each Intersection**

This step is based on the Range Network Expansion (*RNE*) technique. For each segment on the path we do the following processing to calculate the network distance between the objects and the intersections on the query path. We expand both ends of the segment by applying *RNE* to retrieve all the interest entities within a network distance less than $e$, and add them to the pre-result list *PRL*. The objects ($p8$, $p9$ and $p14$) that passed the last steps will be excluded in this step because their network distance to the query path is not less than $e$ (see Table 6.1), and in the same time their path will also be excluded. The segments of the query path $(S,D)$ will be adjusted, and any segment that does not have any intersection will be merged with its neighbour. For example, after removing the object $p9$ and $p14$, the segments $(r_6,r_7,3)$, $(r_7,r_8,5)$, $(r_8,r_9,3)$ will be combined together in one segment $(r_6,r_9,11)$ (see Figure 6.5). Then, the pre-result list will be $PRL = \{p1,p2,p3,p4,p5,p6,p7\}$, $\forall pi \in PRL, (D_N(pi,q) < e)$, where $q$ is the query path $(S,D)$, and the network distance between the objects and the ends of the segments will be added to the expansion list *EL*.

### 6.3.3 Half Space Step - on Original Path

The half space method will be applied on the query path to consider which object represents the forward direction of the moving query. For each object in the *PRL*, we will draw a perpendicular line to the path query that goes through the intersection between the query path $[S,D]$ and the path that will lead to the object. Considering the perpendicular line as the $y$ axis, forward half space will include the first and fourth quadrants (the $x$ axis is positive), while the backward half space will include the second and third quadrants (the $x$ axis is negative). Each object is considered as backward will be eliminated from

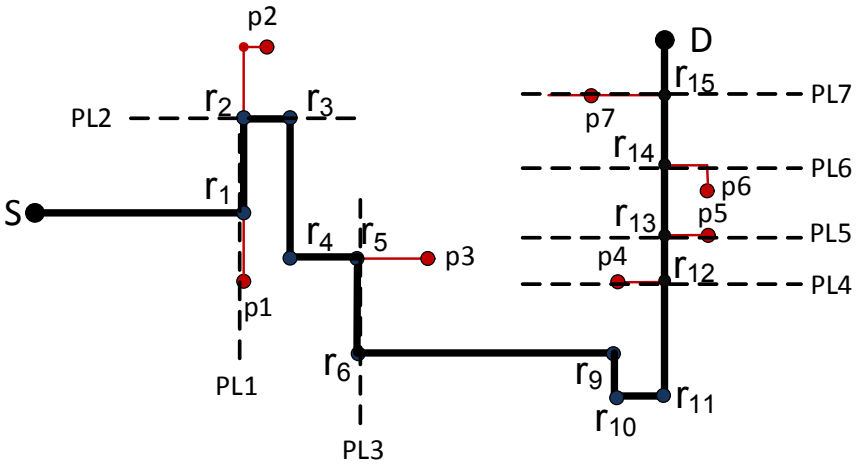



Figure 6.6: Half space step

the *PRL*. Figure 6.6 shows the perpendicular lines that are drawn in a dished line, where *PLi* represents the half space of object *pi*. In this figure, the objects $\{p1, p2, p3, p4, p5, p7\}$ and $\{p6\}$ are forward and backward objects respectively. The backward object(s) will be pruned from the *PRL*.

### 6.3.4 Split Step

The fifth step is the split step, which is used to determine at which point the user will obtain a new result. This result either has a new interest object that enters the range search, or an object becomes out-of-date (i.e., leaves the range search) and should be removed from the result. In either way, the communication between the user and the database server should be reestablished at the split point.

All interest objects in the pre-result list that pass the last four steps should get through the split step. In this step, two split points (*si*, *di*) are determined on the query path for each object of interest *pi* within the range search. The two split points (*si*, *di*) create a subsegment on the query path, where (*si*, *di*) represent the start and the end of the subsegment respectively. At each splitting

point, the result should be updated when an object enters or leaves the range. For each split point $si, di \in SP$ ($1 \leq i \leq n$, where $n$ represents the number of interest objects in *PRL*): $si, di \in$ path $[S, D]$, all points in subsegment ($si$, $di$) will consider $pi$ as one of the interest objects, where $SP$ is the split points list and *PRL* is the pre-result list which contains all objects of interest within the range search from the query path that passed the previous steps. The split step determines two split points for each interest object in the *PRL* by drawing and calculating the position of the split nodes using the formula of:

$$SP[pi] = e - D_N(Y, pi) \tag{6.1}$$

where $Y$ is the end of the segment, $pi$ is an object in the pre-result list and $e$ is the radius of the query. However, because we are only concerned with the forward objects, then the split point when the object leaves the range search ($di$) will be the intersection between the query path and the path that leads to the object $pi$ to prevent the query from diverting back to reach the query.

Figure 6.7 shows how the split point will be created on the query path. For each object in the pre-result list, two split points will be located on the query path at ($si$, $di$). $\forall si, di \in SP\{s1, s2, ..., sn, d1, d2, ..., dn\}$, $D_N(S, si) < D_N(S, di)$: $1 \leq i \leq n$ , where $n$ represents the number of interest objects in *PRL*, and $si$ and $di$ represent the location where the object $pi$ enters and leaves the range search, respectively (i.e., ($s1$, $d1$) are the split points of $p1$, ($s2$, $d2$) are the split points of $p2$ and so on). To avoid scanning the database repeatedly, all split points with their corresponding coverings are reported. The split point list $SP$ has the start point $S$ if there are no objects left in the range since the start.

Figure 6.7 shows overall there are fourteen split points: two split points at distance 7 and 9 from $S$ at segment $Sr_1$, two split points at distance 3, and 4

from $r_1$ at segment $r_1r_2$, two split points at distance 1 and 3 from $r_4$ at segment $r_4r_5$, three split points at distance 2, 4 and 5 from $r_{11}$ at segment $r_{11}r_{12}$, one split point at distance 2 from $r_{12}$ at segment $r_{12}r_{13}$ and lastly, two split points at distance 4 and 6 from $r_{13}$ at segment $r_{13}r_{15}$. The query results for segment $Sr_1$ are:

| | |
|---|---|
| $S = \{\phi\}$ | no object within the range query at the beginning |
| $S \rightarrow s1 = \{p1\}$ | $p1$ is entering at $s1$ |
| $s1 \rightarrow r_1 = \{\phi\}$ | $p1$ is leaving at $r_1$ |

While the query results for segment $r_1r_2$ are:

| | |
|---|---|
| $r_1 = \{\phi\}$ | |
| $r_1 \rightarrow s2 = \{p2\}$ | $p2$ is entering at $s2$ |
| $s2 \rightarrow r_2 = \{\phi\}$ | $p2$ is leaving at $r_2$ |

The query result will be the same along the segments $r_2r_3$ and $r_3r_4$ which is $\{\phi\}$. The query results for segment $r_4r_5$ are:

| | |
|---|---|
| $r_4 = \{\phi\}$ | |
| $r_4 \rightarrow s3 = \{p3\}$ | $p3$ is entering at $s3$ |
| $r_5 = \{\phi\}$ | $p3$ is leaving at $r_5$ |

The query result will be the same along the segments $r_5r_6$, $r_6r_9$, $r_9r_{10}$ and $r_{10}r_{11}$ which is $\{\phi\}$. While the query results for segment $r_{11}r_{12}$ are:

$r_{11} = \{\phi\}$

$r_{11} \rightarrow s4 = \{p4\}$ $p4$ is entering at $s4$

$s4 \rightarrow s5 = \{p4, p5\}$ $p5$ is entering at $s5$

$s5 \rightarrow r_{12} = \{p5\}$ $p4$ is leaving at $r_{12}$

While the query results for segment $r_{12}r_{13}$ are:

$r_{12} = \{p5\}$

$r_{12} \rightarrow r_{13} = \{\phi\}$ $p5$ is leaving at $r_{13}$

While the query results for segment $r_{13}r_{15}$ are:

$r_{13} = \{\phi\}$

$r_{13} \rightarrow s7 = \{p7\}$ $p7$ is entering at $s7$

$s7 \rightarrow r_{15} = \{\phi\}$ $p7$ is leaving at $r_{15}$

Finally, the query result will be the same along the segments $r_{15}D$ which is $\{\phi\}$.

According to Figure 6.7, the query path $[S, D]$ is divided into a number of segments; they are the link between consecutive intersections. In addition, there are two types of subsegments created on the query path, namely, object segment and result segment. The object segment is created by an interest object in the location where it enters and leaves the range search. For each object segment, $si$ and $di$ are the start and the end point of this segment respectively, where $si, di \in SP$ and $pi \in PRL$. For example, within the object segment $(s1, r_1)$, $p1$ will remain in the result list from $s1$ to $r_1$, even when the moving query passes any number of split points. Hence, $p1$ will be safe within this segment. On the other hand, the result segment which is created by either intersection between

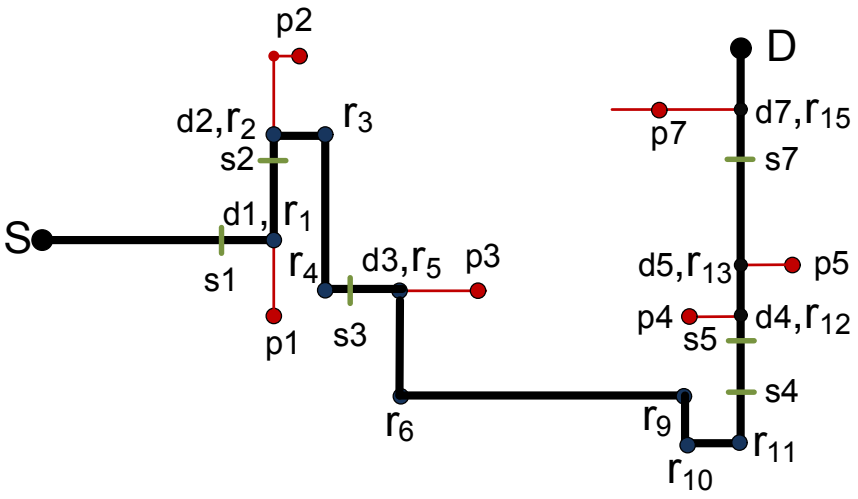



Figure 6.7: Split step

two object segments (i.e., $(s4, s5)$), or the gap between an object segment and its neighbour (i.e., $r_{13}, s7$), will remain unchanged while the moving query is within the result segment, because no object is entering or leaving the range. Consequently, at each split point the result can only be changed.

Let's take Figure 6.8 to show the three types of segments in more detail. There are four road segments $(r_{11}, r_{12})$, $(r_{12}, r_{13})$, $(r_{13}, r_{15})$ and $(r_{15}, D)$. While the subsegments $(r_{11}, s4)$, $(s4, s5)$, $(s5, r_{12})$, $(r_{12}, r_{13})$, $(r_{13}, s7)$, $(s7, r_{15})$ and $(r_{15}, D)$ represent the result segments. On the other hand, the subsegments $(s4, r_{12})$, $(s5, r_{13})$ and $(s7, r_{15})$ represent the object segments. $p4$ is the only result of the query while the query moves in the result segment ($s4$, $s5$). When the query enters the result segment $(s5, r_{12})$ then the result will be ($p4$, $p5$). With the continuous movement of the user, reaching the result segment $(r_{12}, r_{13})$ will make the result become ($p5$). It is very obvious that the object $p4$ remains in the result list while the query is moving within the object segment $(s4, r_{12})$, and $p5$ remains in the result while the query is in the object segment $(s5, r_{13})$

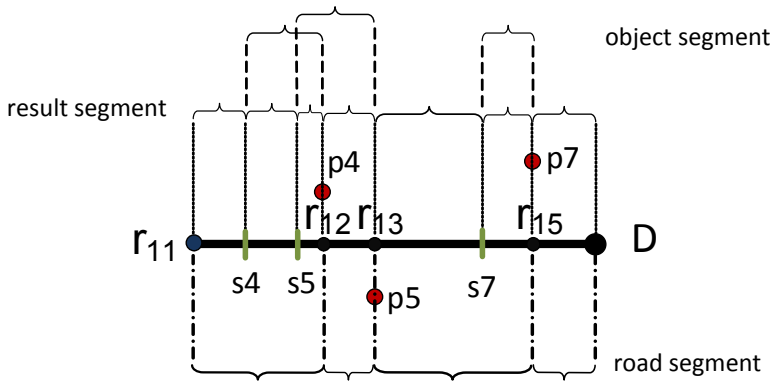



Figure 6.8: Object segment and result segment

and so on. Therefore, after finding out all split points, the communication between the user and the database server will occur only when the user reaches a split point to update the result list.

### 6.3.5 Result Step

Up to this step we have two lists, pre-result list and split points list. The pre-result list will be given to the user step by step according to the split points in the split points list.

### 6.3.6 LMR Algorithm

R-tree is used to index the objects with high performance. A query path $[S,D]$ and the radius of the query $e$ are given to find all objects of interest, such as post offices, within this radius along the query path. This method will consider Euclidian and network distance to collect the objects of interest.

The algorithm of the lookforward moving range query based on the road network works as follows: each intermediate node, $R$, should be determined as a candidate and then to be expanded if $MinDist(R,q) < e$. That means, this internal node which is within the range search of $e$ should be expanded to

**Algorithm 6.1.** Lookforward moving range query algorithm

```
1: /* [S,D]: the query path (q) and e: the radius of the query */
2: /* Find all objects around the query q which are located in the forward
   direction */
3: LIST RL, SP, FL, PRL
4: Start from the root of the R-tree
5: Find all candidates MBR R, MinDist(R,q) < e
6: for each candidate MBR do
7:    if D_E(pi,q) < e then
8:       FL.add(pi)
9:    end if
10: end for
11: divide the query path [S,D] to segments
12: each segment has two ends (r_j, r_{j+1})
13: for each segments in the query path do
14:    expand r_j, r_{j+1} in all directions for D_N(e)
15:    each object (pi) in FL,
16:    if D_N(pi, r_j(r_{j+1})) < e then
17:       PRL.add(pi)
18:    end if
19: end for
20: for each intersection r_j in [S,D] do
21:    draw a perpendicular line at it
22:    if pi is backward then
23:       PRL.remove(pi)
24:    end if
25: end for
26: for each object pi in PRL do
27:    find two split points, si and di, on the query path q, where D_N(pi,si) = e
       and D_N(pi,di) = D_N(pi,q)
28:    SP.add(si)
29:    SP.add(di)
30:    RL.add(pi)
31:    RL.add(si)
32:    RL.add(di)
33: end for
```

look for qualified object(s). The algorithm will determine whether the qualified object $pi$ from a candidate internal node has $D_E(pi,q) < e$, and then will place this object in the filter list $FL$.

All qualified objects in $FL$ within Euclidean distance less than $e$ from the query path $[S,D]$, should be tested against the network distance in order to find the objects of interest. Each segment in the query path will be examined, and the expansion from both ends of the segment will occur using the Range Network Expansion (RNE) technique. Only the direction which has an object in $FL$ will be examined to find out which object falls within network distance $e$ from $[S,D]$ and this object will added to *PRL*.

For each object in *PRL*, two split points will be determined: first *si* when this object becomes within the range search $e$ from $[S,D]$ (joins the query), and second is the split point *di* when this object becomes behind the query (expires). For each split point, joining or expiring, the database server should send up-to-date results to the user.

## 6.4 Simplifying the Query Path

The half space step (in Section 6.3.3) will be applied to the original path (path query). Some objects may represent the forward objects according to the current query location, but by taking the whole path they may not. When there is a small bend, a u-turn or zig zag, in the road the result will not be accurate, this happens when the half space depends on the original path. For example $p1$ in Figure 6.6 should not be considered as forward objects because the query will change its direction. Also, finding the perpendicular line on a non straight line is complicated (i.e., the data of the query path comes from raster datasets) (ESRI, 2014). Therefore, we developed two other methods to find the forward objects, named simple trajectory and approximate trajectory.

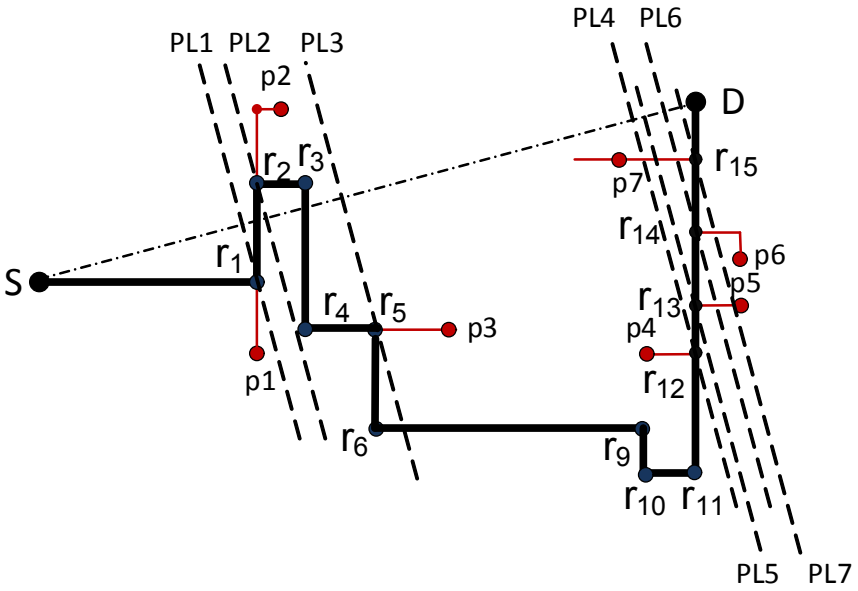



Figure 6.9: Simple approximate trajectory

### 6.4.1 Simple Trajectory

In this method, a straight line (called simple trajectory) is drawn between the start point $S$ and the destination point $D$ ($\overline{SD}$) of the original path (the query path). Then, a perpendicular line is drawn from each intersection located on the original path to the simple trajectory $\overline{SD}$. Figure 6.9 shows the simple trajectory method. By applying this method, the objects $\{p2, p3, p5, p6\}$ are forward objects while $\{p1, p4, p7\}$ are backward objects. This method solves the problems that faced the original path, which were mentioned in the previous section, but it fails to give accurate results. For example, it includes $p6$ as a forward object.

### 6.4.2 Approximate Trajectory

We have adopted the Douglas Peucker (DP) algorithm to find more accurate lookforward objects. The reason for choosing the batch algorithm (e.g., DP) is that the location server is able to access the whole data series of the path query.

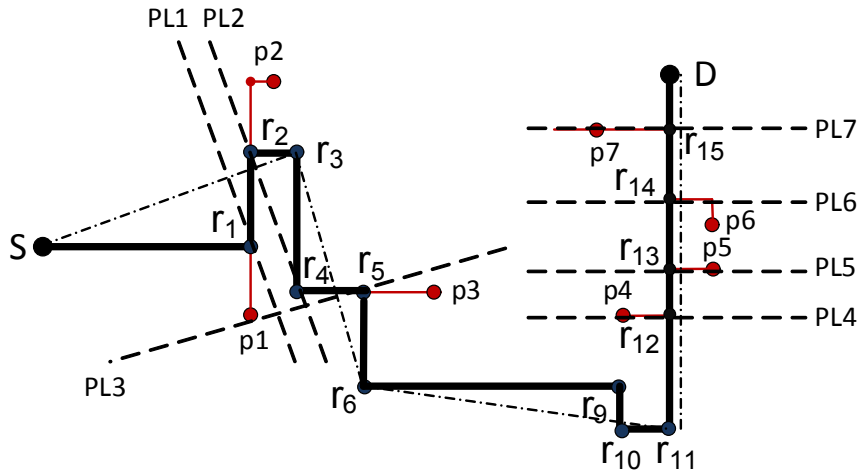



Figure 6.10: Approximate trajectory using Douglas Peucker algorithm

DP considers the original path as time segments; in our work the path is the intersection segments. After finding the approximate trajectory, a perpendicular line between the intersection and the corresponding approximate segment will be applied. Figure 6.10 shows our method on the approximate trajectory to find the forward objects. The objects $\{p2, p3, p4, p5, p7\}$ are forward objects while $\{p1, p6\}$ are backward objects. The result of this method is more genuine than the other two methods.

## 6.5 Experimental Results

To examine our approach, the lookforward moving range query, we use the Melbourne City road network provided by google maps *https://maps.google.com.au/* and ArcGIS maps *http://www.arcgis.com/home/webmap/viewer.html*.

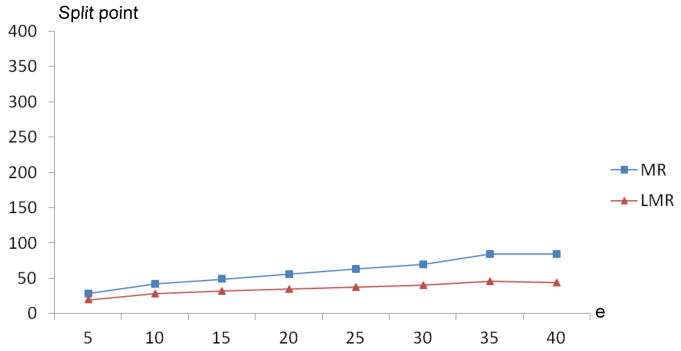
### 6.5.1 Experiment

We measure the performance of our LMR technique with respect to the average number of query location updates compared with the moving range query. Note that, query location update means that at each split point the query will establish a communication with the database server to send its new location, and at the same time the server will send back an updated result list to the query. By reducing the number of split points the query will get a better performance.

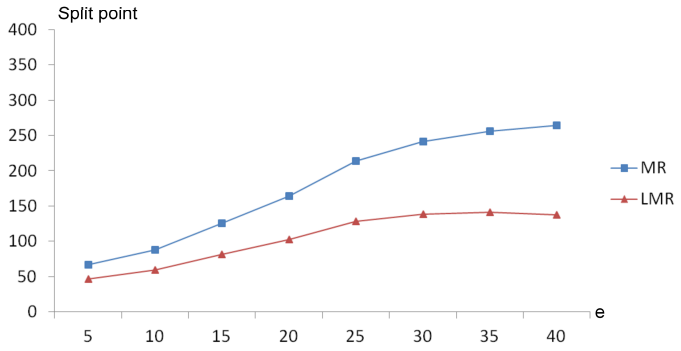
We select 200 random queries in various distributions and range sizes. To create the average result, 100 objects, 300 objects and 500 objects represent the low, medium and high density environments respectively. We consider the post offices as low density objects, petrol stations as medium density objects and restaurants (including takeaways) as high density objects. In addition, we use different range searches, from 5$km$ to 40$km$.

Figure 6.11 shows the average result of the experiments that examine the number of split points in the lookforward moving range query approach and compare it with the moving range query. The experiment shows, regardless of the length of the query's radius and the number of objects in the dataset, the number of split points are within a linear growth in both of the techniques.

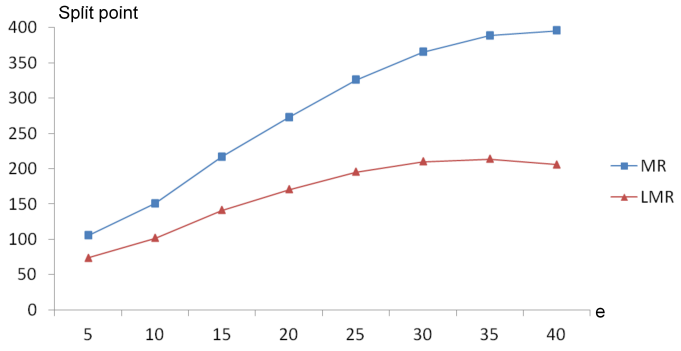
Considering the total number of split points (updating load), the total is considerably large in the moving range query compared with LMR. In the low (Figure 6.11(a)), medium (Figure 6.11(b)) and high (Figure 6.11(c)) density figures the number of split points of our technique is approximately 30% fewer than in the moving range query when the radius of the query is small (e.g, 5$km$). Whereas, the number of split points in the LMR query is quite less than in the moving range query when the radius of the search becomes bigger (e.g, when the radius is 40$km$, it is approximately a half). On the other hand, the




(a) low density 100 objects



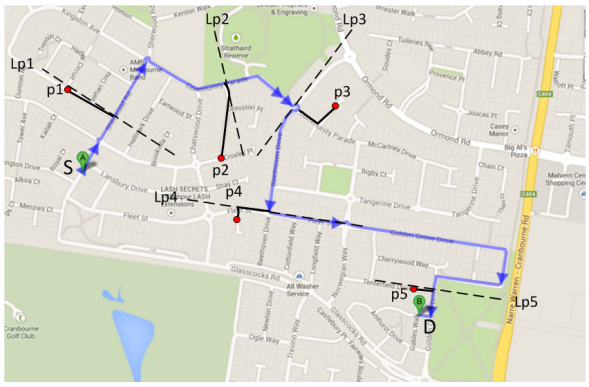

(b) medium density 300 objects



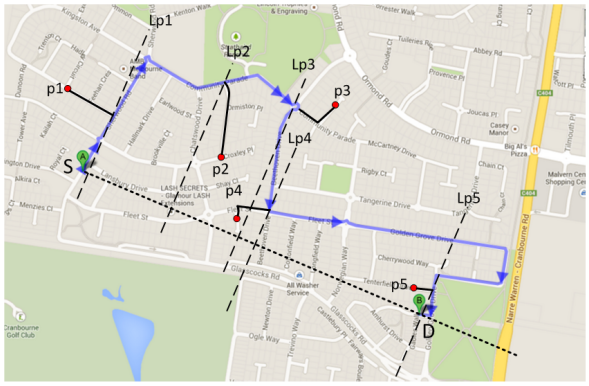

(c) high density 500 objects

Figure 6.11: Number of split points in moving range query and LMR based on road network

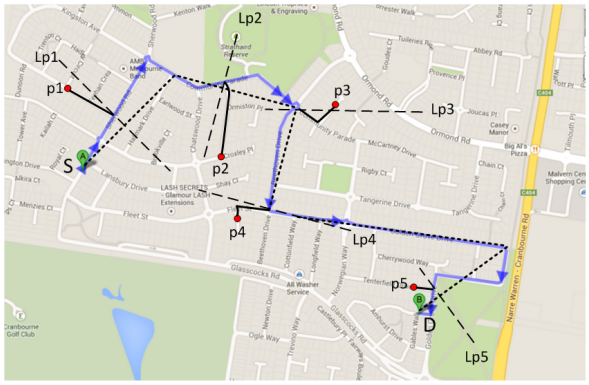
number of false hits in our technique is slightly less than in the moving range query, because we eliminate some *MBRs* in the filter step.




(a) Using Original path




(b) Using simple trajectory




(c) Using approximate trajectory

Figure 6.12: Case study 1

### 6.5.2 Case Studies

In this section we present two case studies to explain our methods (original path, simple trajectory and approximate trajectory) to find the forward objects.

**Case 1**

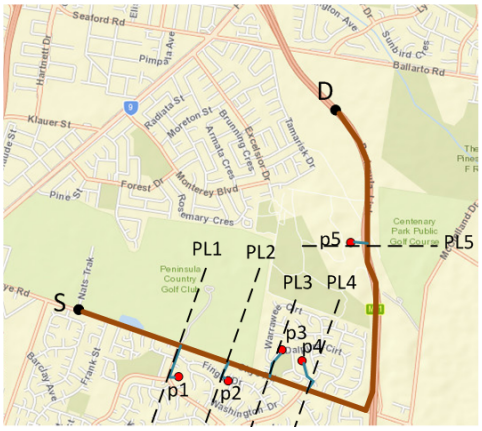
Figure 6.12 shows our three methods, original path (Figure 6.12(a)), simple trajectory (Figure 6.12(b)) and approximate trajectory (Figure 6.12(c)). As shown in these figures, $p3$ is considered as a backward object only in the approximate trajectory method, which is correct. On the other hand, $p4$ is considered as a backward object only in the simple trajectory method and that is correct, but at the same time this method has failed to consider $p5$ as a forward object.

**Case 2**

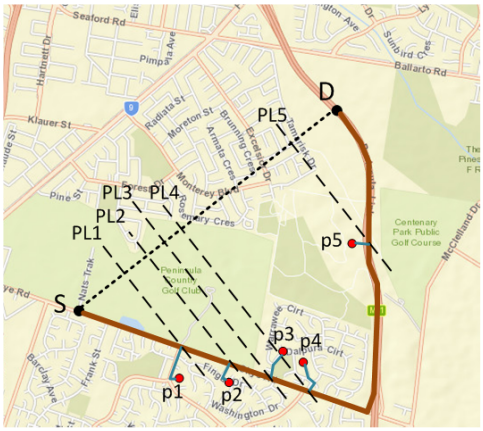
Figure 6.13 shows our three methods, original path (Figure 6.13(a)), simple trajectory (Figure 6.13(b)) and approximate trajectory (Figure 6.13(c)). As shown in these figures, the simple trajectory method considers $p4$ as a forward object, which is false. Moreover, this method considers $p5$ as a backward object which is also false. On the other hand the results of the original path method and the approximate trajectory method are matched.

## 6.6 Summary

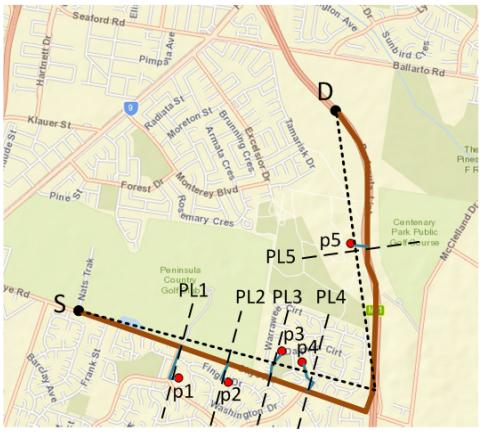
In this chapter, we present the lookforward moving range query based on the spatial road network. Our technique relates Euclidean and network distance to retrieve related objects and to exclude unrelated ones. We have adapted the Range Network Expansion (RNE) technique to expand the road network on selected paths looking for objects of interest. When the server calculates objects of interest, each object will be examined whether it is forward relate to the user or not. For each object of interest that the user considers as a backward object, the server will eliminate it and will not send it to the user. We introduce three




(a) Using Original path




(b) Using simple trajectory




(c) Using approximate trajectory

Figure 6.13: Case study 2

different methods to achieve this task. All three methods introduce some error. However, the approximate trajectory method is the most accurate method. This is an improvement on giving the user all results because: *i)* the user needs some time in advance to plan and react to the result, *ii)* the high number of updates between the user and database server, will increase the overall workload on the server, and *iii)* the users may not be willing to divert from their route, when it takes them far from their original path.

# Chapter 7

## Conclusion and Future Work

# 7

# Conclusion and Future Work

## 7.1 Overview

In this thesis, we have presented efficient techniques to deal with the moving range query based on Euclidean and network distances under different settings. Chapter 3 presented some variants of approximate range queries in Section 3.2, Section 3.3, Section 3.4 and Section 3.5 respectively. Chapter 4 presented the novel types of dynamic safe regions for moving range queries. Three different approaches were discussed in Section 4.2. In addition, a novel method of calculating the area of safe regions was presented in Section 4.3. Chapter 5 introduced a novel method of monitoring queries that move arbitrarily. Chapter 6, presented three different methods of finding lookforward objects in Section 6.3 and Section 6.4. Finally, the contribution of the thesis and future work are outlined in this chapter.

## 7.2 Conclusion

This thesis addresses the challenges of support for querying, updating and monitoring in moving query environments. Various techniques to efficiently answer different types of moving range query based on Euclidean and network distance under different circumstances were presented. Chapter 3 outlined our research on efficient approximate results of static and moving queries based on Euclidean and road network databases. In Chapter 4 and Chapter 5, new types of dynamic safe regions for the moving range query were presented. A unique technique to monitor moving queries to minimise the computation and communication costs was also introduced. In Chapter 6, our approach for monitoring a special type of moving range query, which emphasises objects in the direction of the moving query was illustrated. These achievements are detailed below.

In Chapter 3, we addressed the drawbacks of the moving range query under different circumstances, and introduced different new approaches namely, (ASR and AMR based on Euclidean distance) and (*ARER*, *ARNE* and AMR based on a spatial road network) to obtain the query result effectively. The main idea behind our approaches is to use the advantages of the approximation in order to improve the performance of range queries. Therefore, our new queries obtain results faster, with less computation costs. The new queries are aimed at reducing the critical objects that enter and exit a range search over a short time interval, thus giving a user time to make a decision. By reducing the number of false hits and number of communications with the server, these queries reduce the number of split points. Several experiments on each approach were conducted. The results show that our proposed queries give a better performance in terms of search time and search accuracy.

Chapter 4, we focussed on two main goals: first, minimising the frequent updates of the query location; and second, keeping costs low while monitoring the moving query. Consequently, we proposed a continuous range safe region methods for mobile navigation with moving queries. We demonstrated that our methods avoid supplementary communication between query and server while the query is in the safe region which reduces the need for continuous query monitoring and eliminate need for a user to follow a defined path, and allows the user to avoid location disclosure while in the safe region thereby retaining privacy. Evaluation of the performance of our methods shows that our method reduces server query monitoring and reduces communication costs while the query moves within its safe region.

In Chapter 5, we discussed the fact that predicting the direction of the queries becomes a challenge to the research community because the movement of the queries cause a query's results to change continuously. We address this challenge in this chapter and propose a novel method to monitor the position of the query over time using a linear function based on the direction of the query obtained by periodic monitoring of its position. Our method reduces the load on the server by making the client monitor itself; on top of that, the client can have more privacy while moving. We use the time and the concept of the safe region together, hence, if the query makes a sudden turn, the result will not be affected because the query will still be located inside the safe region. In our method, the user will have an indication about how far away the nearest object to the query is, and in what direction from the query.

Finally in Chapter 6, we proposed a novel query processing technique for the moving range query, called lookforward moving range query. In spatial databases, all traditional range queries try to find all objects of interest around

the query point without controlling the query direction. By contrast, we introduce a new way of distinguishing between significant and minor important objects within the boundary of the moving range query. We only attend to objects located in the direction of the moving user and exclude objects behind the moving user or those that will cause a change in trajectory. We also improve the filter step to reduce the number of candidate objects and the number of communications with the server. The resulting technique delivers a better running time and performance compared with its competitors.

## 7.3 Future Work

There are several possible directions for future work in which our ideas can be applied directly to other applications, or our research can be extended to cover other areas.

Incomplete and uncertain data are a fundamental problem in spatio-temporal databases. The reasons for making the data incomplete and uncertain are: *i*) inaccurate prediction for moving objects, *ii*) the power outage of the mobile device, and *iii*) the errors during communication between the moving objects and the spatio-temporal database server; due to the wireless working nature of the mobile device (García-Laencina et al., 2009; W. Cheng et al., 2009). Our approximate query can work efficiently with these types of data. Also, a safe zone can be implemented to surround similar data to subsidise the missing values.

Aggregation is another important area of study in today's relational database management systems. Much research has been conducted to improve the performance of the aggregation operations in spatial databases. Some of them have concentrated on improving the spatial indexing techniques, while others have

focussed on improving the computational algorithms of the aggregation queries. Many delicate indexing techniques have been developed to answer range aggregate queries efficiently; the most popular one is the aggregation R-tree (AR-tree) (Yang et al., 2009). However, AR-tree suffers from some limitations: First, a lot of information has to be read from the disk. In addition to reading a large deal of irrelevant information to the query for partially-dimensional range queries (Feng & Makinouchi, 2010). Second, when any object in the tree is deleted, inserted or updated; the whole aggregate function on one or more subtree(s) should be recalculated. Finally, since each intermediate node keeps the aggregate information, the capacity of the nodes will decrease and disk access will increase (i.e., extra bytes for aggregate information). Therefore, these problems make obtaining the exact solutions time consuming and difficult to implement. Consequently, we will consider these problems by developing an approximate model to process aggregate queries using R-tree. We are expecting to overcome the limitations of range aggregates that use AR-tree by providing acceptable approximate answers within a short response time.

Regarding queries, we will consider more query examinations using various features, such as different types of roads, different geometric areas, and different underlying networks. We also intend to work on other spatial query types such as (nearest neighbours query, join query, point query and reverse nearest neighbours query). Each has a set of unique features. Moreover, we are interested in including random moving query to the lookforward objects method, and using the sliding window algorithm (explained in Section 6.2) to approximate the query path when the users move randomly or do not want to reveal their path to the server.

Regarding the updating process, we can further consider the algorithms that support moving queries inside different sizes and shapes of safe regions (i.e., buildings). The concept of a safe region can be applied on indoor spaces too.

Further research with respect to monitoring will generalise our method in multi-environments. Prior (History) is also useful to predict the next query's move. Finally, non-linear functions (i.e., recursive motion function (Tao et al., 2004)) may be used to find a curve that fits the last few reported locations of a moving query.

# Appendix A

# Abbreviations

Table A.1: Symbols used throughout this thesis

| *Symbols* | *Definition* |
|---|---|
| $d$ | distance units |
| $di$ | split point when object $pi$ expired |
| $e$ | radius of range query |
| $e'$ | radius of basic safe region |
| $e''$ | radius of enhanced safe region |
| $\varepsilon$ | distance absolute error |
| $lowerbound$ | $e \times (1-\gamma)$ |
| $n$ | number of objects |
| $ni$ | intersection node |
| $\rho$ | distance ratio error ($e \times \gamma$) |
| $pi$ | $pi \in P$ |
| $q$ | query point |
| $r_a$ | anchor point |
| $r_f$ | float point |

*Continued on next page*

Table A.1 – *Cont.*

| *Symbols* | *Definition* |
|---|---|
| $si$ | split point when object $pi$ joins the range |
| $t$ | time unit |
| $upperbound$ | $e \times (1+\gamma)$ |
| $v$ | velocity of the query |
| $\gamma$ | distance relative error or error threshold |
| AMR | approximate moving range query |
| ARER | Approximate Range Euclidean Restriction |
| ARNE | Approximate Range Network Expansion |
| ASR | approximate static range query |
| BOPW | Before Opening Window |
| $C$ | candidate objects |
| CPU | central processing unit |
| $D_E$ | Euclidean distance |
| $D_N$ | network distance |
| DP | Douglas Peucker algorithm |
| $EL$ | expansion list |
| $F_d$ | minimum distance between where the object falls just inside the range and the boundary of the range |
| $FL$ | filter list |
| I/O | input/output |
| GIS | Geographic Information Systems |
| GPS | Global Position System |
| HD | high density (500 Objects) |

*Continued on next page*

Table A.1 – *Cont.*

| *Symbols* | *Definition* |
|---|---|
| LD | low density (100 Objects) |
| LMR | lookforward moving range query |
| MBR | Minimum Boundary Rectangle |
| MD | medium density (300 Objects) |
| $MinDist(q,x)$ | minimum distance between query $q$ and $x$ ($x$ is either an object or a minimum boundary rectangle) |
| $N$ | set of intersection nodes |
| $N_d$ | minimum distance between where the object falls just outside the range and the boundary of the range |
| NOPW | Normal Opening Window |
| $P$ | set of objects of interest |
| $PL_i$ | perpendicular line on intersection which leads to object $pi$ |
| $PRL$ | pre-result list |
| $QR$ | queue result list |
| $QS$ | qualifying segments |
| $R$ | non leaf node |
| $RL$ | result list |
| RNE | Range Network Expansion |
| RER | Range Euclidean Restriction |
| $RP$ | set of range objects |
| $Rpi$ | range of $pi$ |
| $S,D$ | start and destination points for a query path |

*Continued on next page*

Table A.1 – *Cont.*

| *Symbols* | *Definition* |
|---|---|
| $SmDist(N_d,F_d)$ | smallest amount between $N_d$ and $F_d$ |
| $SP$ | split points list |
| $SOL$ | safe-object list |
| $X$ | R-tree node |

# Publications

Publications arising from this thesis include:

1. AL-Khalidi, H. Taniar, D. and Safar, M. (2011), Approximate static and continuous range search in mobile navigation. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*. ICUIMC'11, ACM, pp.1-10.

2. AL-Khalidi, H. Taniar, D. and Safar, M. (2013), Approximate algorithms for static and continuous range queries in mobile navigation. In *Computing*. Springer, 95(10-11), pp.949-976.

3. AL-Khalidi, H. Abbas, Z. and Safar, M. (2013), Approximate range query processing in spatial network databases. In *Multimedia Systems*. Springer, 19(2), pp.151-161.

4. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), On finding safe regions for moving range queries. In *Mathematical and Computer Modelling*. Elsevier, 58(5-6), pp.1449-1458.

5. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), Dynamic safe regions for moving range queries in mobile navigation. In *International Journal of Ad Hoc and Ubiquitous Computing*. (Accepted)

6. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2013), Efficient monitoring of moving mobile device range queries using dynamic safe regions. In *The 11th International Conference on Advances in Mobile Computing and Multimedia*. MoMM'13.

7. AL-Khalidi, H. Taniar, D. Betts, J. and Alamri, S. (2014), Monitoring moving queries inside a safe region. In *The Scientific World Journal*. Hindawi.

8. Alamri, S. Taniar, D. AL-Khalidi, H. and Nguyen, K. (2014), Density-based for moving query in multi-floor indoor spaces. IEEE Transactions on Knowledge and Data Engineering. (Under Review).

9. AL-Khalidi, H. Taniar, D. Nguyen, K. Betts, J. and Alamri, S. (2014), Lookforward moving range search query based on road network. In *Computers & Mathematics with Applications*. Elsevier. (Under Review)

Permanent Address: Clayton School of Information Technology

Monash University

Australia

This thesis was typeset with LaTeX $2\varepsilon$[1] by the author.

[1] LaTeX $2\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

# References

Ahmed, S., & Kanhere, S. S. (2012). On the characterisation of vehicular mobility in a large-scale public transport network. *International Journal of Ad Hoc and Ubiquitous Computing*, *11*(2/3), 68–81.

Alamri, S., Taniar, D., & Safar, M. (2013). Indexing moving objects for directions and velocities queries. *Information Systems Frontiers*, *15*(2), 235-248.

Alamri, S., Taniar, D., Safar, M., & Al-Khalidi, H. (2013). Spatiotemporal indexing for moving objects in an indoor cellular space. *Neurocomputing*, *122*(0), 70 - 78.

Alavi, B., & Pahlavan, K. (2003). Bandwidth effect on distance error modeling for indoor geolocation. In *Personal, Indoor and Mobile Radio Communications* (Vol. 3, pp. 2198–2202).

AL-Khalidi, H., Abbas, Z., & Safar, M. (2013). Approximate range query processing in spatial network databases. *Multimedia Systems*, *19*(2), 151-161.

Al-Khalidi, H., Taniar, D., Betts, J., & Alamri, S. (2013a). Efficient monitoring of moving mobile device range queries using dynamic safe regions. In *Proceedings of International Conference on Advances in Mobile Computing; Multimedia* (pp. 351:351–351:360). New York, NY, USA: ACM.

Al-Khalidi, H., Taniar, D., Betts, J., & Alamri, S. (2013b). Dynamic Safe Regions for Moving Range Queries in Mobile Navigation. (Accepted in: *International Journal of Ad Hoc and Ubiquitous Computing*)

AL-Khalidi, H., Taniar, D., Betts, J., & Alamri, S. (2013). On finding safe regions for moving range queries. *Mathematical and Computer Modelling*, *58*(56), 1449–1458.

Al-Khalidi, H., Taniar, D., Betts, J., & Alamri, S. (2014). Monitoring moving queries inside a safe region. *The Scientific World Journal*, *2014*. doi: 10.1155/2014/630396

Al-Khalidi, H., Taniar, D., & Safar, M. (2011). Approximate static and continuous range search in mobile navigation. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication* (pp. 1–10). ACM.

AL-Khalidi, H., Taniar, D., & Safar, M. (2013). Approximate algorithms for static and continuous range queries in mobile navigation. *Computing*, *95*(10-11), 949-976.

Arya, S., da Fonseca, G. D., & Mount, D. M. (2012). Optimal area-sensitive bounds for polytope approximation. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry* (pp. 363–372). New York, NY, USA: ACM.

Arya, S., Malamatos, T., & Mount, D. (2009). The effect of corners on the complexity of approximate range searching. *Discrete and Computational Geometry*, *41*(3), 398-443.

Arya, S., & Mount, D. M. (1993). Approximate nearest neighbor queries in fixed dimensions. In *SODA '93: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 271–280). Society for Industrial and Applied Mathematics.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *The Journal of the ACM*, *45*(6), 891–923.

Berg, M. d., Cheong, O., Kreveld, M. v., & Overmars, M. (2008). *Computational geometry: algorithms and applications* (3rd ed.). Santa Clara, CA, USA: Springer-Verlag TELOS.

Bern, M. (1993). Approximate closest-point queries in high dimensions. *Information Processing Letters*, *45*(2), 95–99.

Bustos, B., & Navarro, G. (2009). Improving the space cost of k-nn search in metric spaces by using distance estimators. *Multimedia Tools and Applications*, *41*(2), 215–233.

Cheema, M. A., Brankovic, L., Lin, X., Zhang, W., & Wang, W. (2011). Continuous monitoring of distance-based range queries. *IEEE Transactions on Knowledge and Data Engineering*, *23*(8), 1182–1199.

Cheema, M. A., Lin, X., Zhang, W., & Zhang, Y. (2013). A safe zone based approach for monitoring moving skyline queries. In *Proceedings of the 16th International Conference on Extending Database Technology* (pp. 275–286). New York, NY, USA: ACM.

Cheema, M. A., Zhang, W., Lin, X., Zhang, Y., & Li, X. (2012). Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *The VLDB Journal*, *21*(1), 69–95.

Cheng, R., Lam, K.-Y., Prabhakar, S., & Liang, B. (2007). An efficient location update mechanism for continuous queries over moving objects. *Information Systems*, *32*(4), 593–620.

Cheng, W., Jin, X., & Sun, J.-T. (2009). Probabilistic similarity query on dimension incomplete data. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining* (pp. 81–90). IEEE Computer Society.

Cho, H.-J., Kwon, S. J., & Chung, T.-S. (2013). A safe exit algorithm for continuous nearest neighbor monitoring in road networks. *Mobile Information Systems*, *9*(1), 37-53.

Chow, C.-Y., Mokbel, M. F., Naps, J., & Nath, S. (2009). Approximate evaluation of range nearest neighbor queries with quality guarantee. In *SSTD '09: Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases* (pp. 283–301). Springer-Verlag.

Corral, A., Caadas, J., & Vassilakopoulos, M. (2002). Approximate algorithms for distance-based queries in high-dimensional data spaces using R-trees. In *ADBIS '02: Proceedings of the 6th East European Conference on Advances in Databases and Information Systems* (pp. 163–176). Springer-Verlag.

Corral, A., Manolopoulos, Y., Theodoridis, Y., & Vassilakopoulos, M. (2000). Closest pair queries in spatial databases. In *Proceeding SIGMOD '00 Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Vol. 29, pp. 189–200). ACM.

Corral, A., & Vassilakopoulos, M. (2005). On approximate algorithms for distance-based queries using r-trees. *The Computer Journal*, *48*(2), 220–238.

da Fonseca, G. D., de Figueiredo, C. M. H., de Sá, V. G. P., & Machado, R. (2013). Linear time approximation for dominating sets and independent dominating sets in unit disk graphs. In *Approximation and Online Algorithms* (Vol. 7846, p. 82-92). Springer Berlin Heidelberg.

da Fonseca, G. D., & Mount, D. M. (2010). Approximate range searching: The absolute model. *Computational Geometry*, *43*(4), 434–444.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *10*(2), 112–122.

ESRI. (2014). (ArcGIS Desktop: Release 10.2.2, Redlands, CA: Environmental Systems Research Institute, http://resources.arcgis.com/en/help/main/10.2/)

Feng, Y., & Makinouchi, A. (2010). Indexing for range-aggregation queries on large relational datasets. In *International Journal of Database Theory and Application* (Vol. 3, pp. 1–14).

García-Laencina, P. J., Sancho-Gómez, J.-L., Figueiras-Vidal, A. R., & Verleysen, M. (2009). K nearest neighbours with mutual information for simultaneous classification and missing data imputation. *Neurocomputing*, *72*(7-9), 1483–1493.

Ghadiri, N., Baraani-Dastjerdi, A., Ghasem-Aghaee, N., & Nematbakhsh, M. A. (2011). Optimizing the performance and robustness of type-2 fuzzy group nearest-neighbor queries. *Mobile Information Systems*, *7*(2), 123-145.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the International Conference on Management of Data* (pp. 47–57). ACM.

Hershberger, J., & Snoeyink, J. (1992). Speeding up the douglas-peucker line-simplification algorithm. In *Proceedings of the 5th Iternational Symposium on Spatial Data Handling* (pp. 134–143).

Hsueh, Y.-L., Zimmermann, R., & Ku, W.-S. (2009). Adaptive safe regions for continuous spatial queries over moving objects. In *Database Systems for Advanced Applications* (Vol. 5463, p. 71-76). Springer Berlin / Heidelberg.

Hu, H. (2005). A generic framework for monitoring continuous spatial queries over moving objects. In *Proceedings of the International Conference on Management of Data* (pp. 479–490). ACM.

Ilarri, S., Mena, E., Illarramendi, A., Yus, R., Laka, M., & Marcos, G. (2012). A friendly location-aware system to facilitate the work of technical directors when broadcasting sport events. *Mobile Information Systems*, *8*(1), 17-43.

Jang, M.-H., Kim, S.-W., & Shin, M. (2007). Performance of tpr*-trees for predicting future positions of moving objects in u-cities. In *Proceedings of the 1st KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications* (pp. 841–850). Springer-Verlag.

Jayaputera, J., & Taniar, D. (2005). Data retrieval for location-dependent queries in a multi-cell wireless environment. *Mobile Information Systems*, *1*(2), 91–108.

Jensen, C. S., Lin, D., & Ooi, B. C. (2004). Query and update efficient b+-tree based indexing of moving objects. In *Proceedings of the Thirtieth International Conference on Very Large Databases* (Vol. 4, pp. 768–779). VLDB Endowment.

Jeung, H., Yiu, M. L., Zhou, X., & Jensen, C. S. (2010). Path prediction and predictive range querying in road network databases. *The VLDB Journal*, *19*(4), 585–602.

Lee, S., Chon, Y., & Cha, H. (2013). Smartphone-based indoor pedestrian tracking using geo-magnetic observations. *Mobile Information Systems*, *9*(2), 123-137.

Meratnia, N., & By, R. (2004). Spatiotemporal compression techniques for moving point objects. In *Advances in Database Technology* (Vol. 2992, pp. 765–782). Springer Berlin Heidelberg.

Mokbel, M. F., Xiong, X., & Aref, W. G. (2004). SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceeding SIGMOD '04 Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 623–634). ACM.

Mokbel, M. F., Xiong, X., Hammad, M. A., & Aref, W. G. (2005). Continuous query processing of spatio-temporal data streams in place. *Geoinformatica*, *9*(4), 343–365.

Nguyen, K., & Cao, J. (2012). Top-k data source selection for keyword queries over multiple xml data sources. In *Proceedings of the 22Nd International Conference on Database and Expert Systems Applications* (Vol. 38, pp. 156–175). Sage Publications, Inc.

Ogiela, L., & Ogiela, M. R. (2009). *Cognitive techniques in visual data interpretation* (Vol. 228). Springer.

Okabe, A., Boots, B., & Chiu, K. S. S. N. (2000). *Spatial tessellations: Concepts and applications of Voronoi diagrams* (2nd ed.). New York, USA: Wiley.

Papadias, D., Zhang, J., Mamoulis, N., & Tao, Y. (2003). Query processing in spatial network databases. In *Proceedings of the 29th International Conference on Very Large Databases* (pp. 802–813). VLDB Endowment.

Papadopoulos, S., Wang, L., Yang, Y., Papadias, D., & Karras, P. (2011). Authenticated multistep nearest neighbor search. *Knowledge and Data Engineering, IEEE Transactions on*, *23*(5), 641-654.

Pesti, P., Liu, L., Bamba, B., Iyengar, A., & Weber, M. (2010). Roadtrack: Scaling location updates for mobile clients on road networks with query awareness. *Proceedings of the VLDB Endowment*, *3*(1-2), 1493–1504.

Petkova, A., Hua, K. A., & Aved, A. (2009). Processing approximate moving range queries in mobile sensor environments. In *Proceedings of the 2009 International Conference on Computational Science and Engineering* (Vol. 2, pp. 452–457). IEEE Computer Society.

Philippe Rigaux, A. V., Michel O. Scholl. (2002). *Spatial databases: With application to GIS*. Morgan Kaufmann.

Price, M. (2012). *Mastering ArcGIS* (Fifth ed.). New York: McGraw-Hill Science.

Ripley, B. D. (1987). *Stochastic simulation*. New York, USA: John Wiley & Sons, Inc.

Roussopoulos, N., Kelley, S., & Vincent, F. (1995). Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (pp. 71–79). ACM.

Safar, M. (2005). K nearest neighbor search in navigation systems. *Mobile Information Systems*, *1*(3), 207–224.

Safar, M. (2008). Spatial queries in road networks based on PINE. *Journal of Universal Computer Science*, *14*(4), 590–611.

Safar, M., & Ebrahimi, D. (2006). eDAR algorithm for continuous knn queries based on PINE. *IJITWE*, *1*(4), 1-21.

Shengsheng, W., & Chen, Z. (2011). A dynamic interval based circular safe region algorithm for continuous queries on moving objects. *IJCNS*, *4*(5), 313-322.

Sistla, A. P., Wolfson, O., Chamberlain, S., & Dao, S. (1997). Modeling and querying moving objects. In *Proceedings of the Thirteenth International Conference on Data Engineering* (pp. 422–432). IEEE Computer Society.

Song, Z., & Roussopoulos, N. (2001). K-nearest neighbor search for moving query point. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases* (pp. 79–96). Springer-Verlag.

Stojanovic, D., Papadopoulos, A. N., Predic, B., Djordjevic-Kajan, S., & Nanopoulos, A. (2008). Continuous range monitoring of mobile objects in road networks. *Data and Knowledge Engineering*, *64*(1), 77–100.

Taniar, D., Leung, C. H. C., Rahayu, W., & Goel, S. (2008). *High performance parallel database processing and grid databases*. Wiley Publishing.

Taniar, D., Safar, M., Tran, Q. T., Rahayu, W., & Park, J. H. (2011). Spatial network RNN queries in GIS. *The Computer Journal*, *54*(4), 617–627.

Tao, Y., Faloutsos, C., Papadias, D., & Liu, B. (2004). Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 611–622). ACM.

Tao, Y., Papadias, D., & Shen, Q. (2002). Continuous nearest neighbor search. In *Proceedings of the 28th International Conference on Very Large Databases* (pp. 287–298). VLDB Endowment.

Tran, Q., Taniar, D., & Safar, M. (2009). Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems I* (Vol. 5740, p. 353-372). Springer Berlin / Heidelberg.

Šaltenis, S., Jensen, C. S., Leutenegger, S. T., & Lopez, M. A. (2000). Indexing the positions of continuously moving objects. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 331–342). ACM.

Waluyo, A. B., Srinivasan, B., & Taniar, D. (2004). A taxonomy of broadcast indexing schemes for multi channel data dissemination in mobile databases.

In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications* (Vol. 2, pp. 213–218). IEEE Computer Society.

Wang, H., & Zimmermann, R. (2011). Processing of continuous location-based range queries on moving objects in road networks. *Knowledge and Data Engineering, IEEE Transactions on*, *23*(7), 1065-1078.

Wu, H., & Hsieh, W. (2012). Location-based vehicular moving predictions for wireless communication. *International Journal of Ad Hoc and Ubiquitous Computing*, *10*(4), 197–206.

Xuan, K., Zhao, G., Taniar, D., Rahayu, W., Safar, M., & Srinivasan, B. (2011). Voronoi-based range and continuous range query processing in mobile databases. *Journal of Computer and System Sciences*, *77*(4), 637–651.

Xuan, K., Zhao, G., Taniar, D., Safar, M., & Srinivasan, B. (2011). Constrained range search query processing on road networks. *Concurrency and Computation: Practice and Experience*, *23*(5), 491–504.

Yang, S., Zhang, W., Zhang, Y., & Lin, X. (2009). Probabilistic threshold range aggregate query processing over uncertain data. In *Proceedings of the Joint International Conferences on Advances in Data and Web Management* (pp. 51–62). Springer-Verlag.

Yung, D., Yiu, M. L., & Lo, E. (2012). A safe-exit approach for efficient network-based moving range queries. *Data and Knowledge Engineering*, *72*, 126–147.

Zhang, J., Zhu, M., Papadias, D., Tao, Y., & Lee, D. L. (2003). Location-based spatial queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (pp. 443–454). ACM.

Zhao, G., Xuan, K., & Taniar, D. (2013). Path knn query processing in mobile systems. *IEEE Transactions on Industrial Electronics*, *60*(3), 1099–1107.

Zhao, G., Xuan, K., Taniar, D., Safar, M., & Srinivasan, B. (2014). Time constraint route search over multi-locations. *The Knowledge Engineering Review*, *29*(2), 217–233.

Zheng, Y., & Zhou, X. (2011). *Computing with spatial trajectories*. Springer.

# Last Thing

"*I can't go back to yesterday because I was a different person then.*"

– Lewis Carroll