

# 國立交通大學

網路工程研究所

碩士論文

於行動環境中一個有效率處理  $k$ -NN 的查詢方法

An Efficient Processing Method of  $k$ -NN Queries in  
Mobile Environments



研究生：黃信翰

指導教授：黃俊龍 教授

中華民國九十六年九月

於行動環境中一個有效率處理  $k$ -NN 的查詢方法  
An Efficient Processing Method of  $k$ -NN Queries in  
Mobile Environments

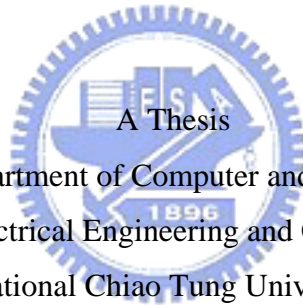
研究生：黃信翰

Student : Hsin-Han Huang

指導教授：黃俊龍

Advisor : Jiun-Long Huang

國立交通大學  
網路工程研究所  
碩士論文



A Thesis

Submitted to Department of Computer and Information Science  
College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Sep. 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年九月

# 國立交通大學

## 博碩士論文全文電子檔著作權授權書

(提供授權人裝訂於紙本論文書名頁之次頁用)

本授權書所授權之學位論文，為本人於國立交通大學網路工程研究所 \_\_\_\_\_ 組， 96 學年度第 1 學期取得碩士學位之論文。

論文題目：於行動環境中一個有效率處理  $k$ -NN 的查詢方法

指導教授：黃俊龍

同意  不同意

本人茲將本著作，以非專屬、無償授權國立交通大學與台灣聯合大學系統圖書館：基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以紙本、光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文全文上載網路公開之範圍及時間：

本校及台灣聯合大學系統區域網路	<input checked="" type="checkbox"/> 中華民國 年 月 日公開
校外網際網路	<input checked="" type="checkbox"/> 中華民國 年 月 日公開

授權人：黃信翰

親筆簽名：\_\_\_\_\_

中華民國 96 年 10 月 8 日

# 國立交通大學

## 博碩士紙本論文著作權授權書

(提供授權人裝訂於全文電子檔授權書之次頁用)

本授權書所授權之學位論文，為本人於國立交通大學網路工程研究所 \_\_\_\_\_  
\_\_\_\_\_組，96 學年度第1學期取得碩士學位之論文。

論文題目：於行動環境中一個有效率處理  $k$ -NN 的查詢方法

指導教授：黃俊龍

### ■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：\_\_\_\_\_，請將論文延至\_\_\_\_\_年\_\_\_\_月\_\_\_\_日再公開。

授權人：黃信翰

親筆簽名：\_\_\_\_\_

中華民國 96 年 10 月 8 日

# 國家圖書館博碩士論文電子檔案上網授權書

ID:GT009456551

本授權書所授權之論文為授權人在國立交通大學資訊學院網路工程研究所  
\_\_\_\_\_組 96學年度第1學期取得碩士學位之論文。

論文題目：於行動環境中一個有效率處理  $k$ -NN 的查詢方法

指導教授：黃俊龍

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

※ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

授權人：黃信翰

親筆簽名：\_\_\_\_\_



民國 96 年 10 月 8 日

1. 本授權書請以黑筆撰寫，並列印二份，其中一份影印裝訂於附錄三之二(博碩士紙本論文著作權授權書)之次頁；另一份於辦理離校時繳交給系所助理，由圖書館彙總寄交國家圖書館。

# 國立交通大學

## 論文口試委員會審定書

本校 資訊科學系 碩士班 \_\_\_\_\_ 君

所提論文：

---


---

---

---

合於碩士資格水準、業經本委員會評審認可。

口試委員：



---

---

---

---

---

---

指導教授：

---

系主任：

中華民國八十年 月 日

# 於行動環境中一個有效率處理 $k$ -NN 的查詢方法

學生：黃信翰

指導教授：黃俊龍

國立交通大學網路工程研究所碩士班

## 摘 要

藉由使用 GPS，人們可以判斷自身的位址並產生其答案與這些位址相關的查詢。如果用戶端於自身儲存體中快取並重複使用這些查詢答案與其相對應的答案正確範圍(valid region)，將有效地降低其等待時間與能源消耗。然而，伺服器基於成本考量並不提供 valid region 予用戶端。因此，利用介於伺服器與客戶端的代理伺服器來提供 valid region 的方法應應而生。但是，這樣的方法仍有效率上的缺失。基於此我們提出改良的方法，並將這個方法延伸應用於  $k$ -NN 查詢上。實驗結果證明我們的方法在縮短整體反應時間與減少網路資料量上有顯著的改良。

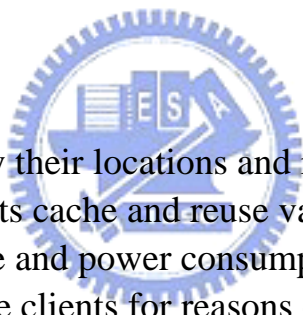
# An Efficient Processing Method of k-NN Queries in Mobile Environments

student : **Hsin-Han Huang**

Advisors : Dr. Jiun-Long Huang

Institute of Network Engineering College of Computer Science  
National Chiao Tung University

## ABSTRACT



Using GPS, people can identify their locations and issue queries whose results depend on those locations. If clients cache and reuse valid regions in their local storage, this reduces response time and power consumption. However, servers may not provide valid regions to mobile clients for reasons of expense. Thus, another method of providing valid regions in proxy between database servers and clients has been proposed. However, this method still suffers from problems of efficiency. We propose an optimized proxy method that can learn valid regions faster than the original method. Moreover, our method can also service k-NN queries that the original cannot. The simulated results show that our method performs better than the existing proxy method in terms of response time and database workload.



## 誌 謝

首先誠摯的感謝指導教授黃俊龍博士，老師悉心的教導使我得以一窺行動管理領域的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

兩年裏的日子，實驗室裏共同的生活點滴，學術上的討論、言不及義的閒扯、讓人又愛又怕的宵夜、趕作業的革命情感、因為睡太晚而遮遮掩掩閃進實驗室.....，感謝眾位學長姐、同學、學弟妹的共同砥礪(墮落?)，你/妳們的陪伴讓兩年的研究生生活變得絢麗多彩。

感謝仕銓學長不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，也感謝承恩、冠翰、瑞男同學的幫忙，恭喜我們順利走過這兩年。實驗室的王禾、建平、國禾學弟、欣怡學妹們當然也不能忘記，你/妳們的幫忙及搞笑我銘感在心。

女朋友逸嫻在背後的默默支持更是我前進的動力，沒有逸嫻的體諒、包容，相信這兩年的生活將是很不一樣的光景。

最後，謹以此文獻給我摯愛的雙親。

# 目 錄

中文提要	.....	i
英文提要	.....	ii
誌謝	.....	iii
目錄	.....	iv
表目錄	.....	vi
圖目錄	.....	vii
一、	Introduction.....	1
1.1	Motivation.....	1
1.2	Challenge.....	2
1.3	Overview of Proposed Approaches.....	3
1.4	Contributions.....	4
1.5	Thesis Organization.....	4
二、	Related Work.....	5
2.1	System Architecture.....	5
2.2	Query Processing.....	5
2.3	Discussions.....	6
2.4	Voronoi Diagrams and Valid Regions.....	7
2.5	Prior Approaches.....	8
2.6	k-Nearest Neighbor Search for Moving Query Points...	11
2.7	Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments.....	11
三、	Query Processing for NN Queries.....	13
3.2	The System Model.....	14
3.2	Overview.....	16
3.3	Organization of the Proxy Cache .....	16
3.4	Cache Lookup.....	16
3.5	Initial EVR Update.....	17
3.5.1	Incremental Right-Hand Algorithm.....	20
3.6	EVR Update.....	22
3.7	Cache Replacement.....	22
3.8	EVR for Client.....	40
四、	Extensions to k-NN Queries.....	26
五、	Simulation.....	29
5.1	Simulation Result.....	29
5.2	Effect of Fast Start.....	29

5.2.1	Effect of proposed EVR Update.....	30
5.2.2	Effect of Using EVR for Client.....	36
5.2.3	Effect of kNN.....	37
六、	Conclusion.....	38



# List of Tables

2.1	Cartogram .....	10
5.1	Simulation parameters .....	31



# List of Figures

1.1	Classification of mobile services .....	2
2.1	System architecture .....	6
2.2	Voronoi diagram .....	8
2.3	Learning by history .....	10
2.4	EVR combination .....	10
2.5	Valid region of 2-NN query .....	11
3.1	The system model .....	14
3.2	Flowchart .....	15
3.3	Organization .....	16
3.4	Fast start .....	20
3.5	EVR update .....	22
3.6	Example of lemma .....	23
3.7	Example of cache replacement .....	25
3.8	Example of client's EVR .....	26
4.1	k-NN queries .....	28
5.1	Area ratio for NN query at very start .....	32
5.2	Proxy hit rate for NN query at very start .....	32
5.3	Waiting time for NN query at very start .....	33
5.4	Average sides for EVR in proxy for NN query at very start .....	33
5.5	Area ratio for NN query .....	34
5.6	Proxy hit rate for NN query .....	34

5.7 Waiting time for NN query ..... 35

5.8 Average sides for EVR in proxy for NN query ..... 35

5.9 Client cache reuse ..... 36

5.10 Waiting time for kNN query ..... 36



# Chapter 1

## Introduction

### 1.1 Motivation

Our daily life has changed because of advances in wireless communication and portable device technologies. Wherever we are, our mobile devices enable us to maintain network connections and request mobile services from mobile service providers. On the other hand, GPS devices allow us not only to determine our geometric coordinates in the plane or in 3-dimensional space, but also to calculate the distance to other positions. Unlike traditional internet services, as a result, clients can request objects or services relative to their location [1].

It will be useful to divide such mobile services into two categories: location-aware services and context-aware services, classified according to the relationship between the query answer and some specific location. To take a simple example of a context-aware service, requests such as querying stock price are time dependent rather than location dependent: information about stock values will be the same wherever the client is located. Querying a ZIP code is another example. On the other hand, queries about both the climate and local traffic are dependent on one's specific location: the result of this kind query will differ according to the user's position. This type of query is called a location-dependent query (abbreviated as LDQ) [12].

Among the queries whose answers depend on specific locations, range query and nearest neighbor query (abbreviated as NN) are the most popular. A range query can be defined as a request referring to the data objects within a specific region. For example, a client might ask a database server for a list of restaurants on the campus at National Chiao Tung University. But this kind of query is irrelevant

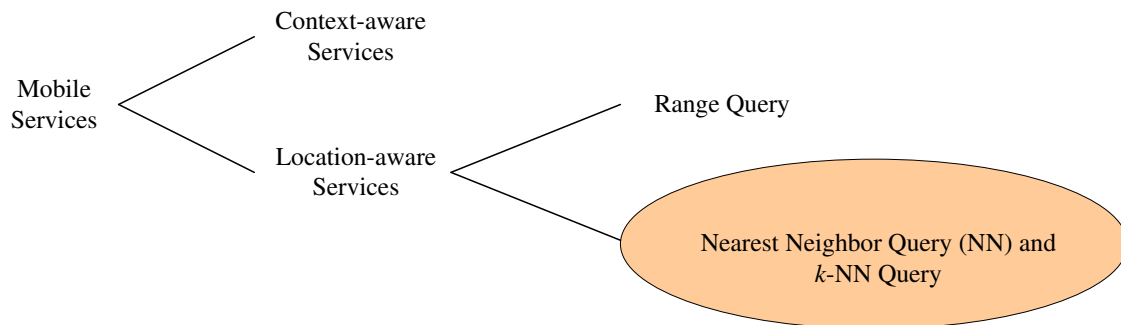
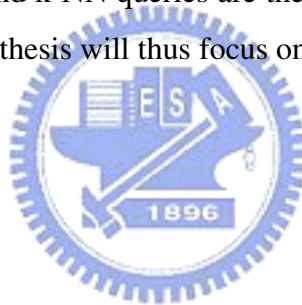


Figure 1.1: Classification of mobile services

to our main subject. We define a NN query as a query regarding the data objects nearest to the client. For example, the user is in a train station, and he wants to know where the nearest restaurant is. In addition to NN query, there is a similar query type called the k-Nearest neighbor query (abbreviated as k-NN). K is the number of nearest data objects which the user is interested in. For example, if the client gets lost while driving downtown, he may request the 3 nearest interchanges that would take him to the freeway. It is clear that NN and k-NN queries are the kind of queries we find in the real world, and offer much for discussion. This thesis will thus focus on NN and k-NN queries. Figure 1.1 shows the classification of mobile services.



## 1.2 Challenge

The question now arises whether, with the increasing number of mobile users, the database server will become the bottleneck that holds back mobile system performance for reasons of limited resources. If so, users will increasingly be the victims of long queues for databases. On the other hand, mobile services attract people because of the small size of mobile devices and the possibility of using various wireless services. Mobile devices continue to get smaller, but the power a small battery can provide is very low; therefore, making maximum use of limited energy is another critical issue.

Region validation is a good solution for such situations [13]. A valid region is a region where the client's solution value is the same. If the client caches a valid region, he may have the chance to reuse this information instead of issuing another query to the server. In this way, the workload of the database and the power consumption of the client is reduced; as a result, the waiting time for each client is shorter.



In the real world, database servers do not provide valid regions to clients. But many researchers have found that the use of proxies to provide valid regions between clients and servers slightly increases the construction cost and workload of each query, but decreases the total workload of the system; therefore, proxy is a good way of providing valid regions to clients [5].

### 1.3 Overview of Proposed Approaches

The proxy server consists of four units: cache hit detector, valid region processor, cache manager and transmitter. When the proxy server receives queries, the cache hit detector first determines which can be solved locally and which should be transmitted to database servers. If the query can be answered by proxy server itself, it is passed to the valid region processor; otherwise, the proxy passes the query to the transmitter. The transmitter encases  $k$ -NN queries as  $(k+1)$ -NN queries, and transmits these new queries to the database servers. On receiving a solution from the database servers, the transmitter decomposes the solution into answers to the original LDQ queries and their corresponding valid regions. Finally, the transmitter gives the answers and valid regions to the valid region processor and cache manager. The cache manager manages cache replacement and updates the cache. The transmitter relays the LDQ answers and the valid regions to the query issuers via base stations. The merge algorithms in our cache manager merge EVR valid regions efficiently and avoid preliminary cache misses. The cache replacement policy algorithm also dell space control. Mobile devices are becoming ever smaller and are powered by batteries which can only deliver a limited amount of power for a limited period. Thus, mobile users do not want a query method which consumes too much energy. The behavior type which costs most in terms of energy for mobile devices is communication with database servers. If a cache is added to the mobile device, the cache stores query answers and valid regions for each query. If users stay within the same valid region area, answers to queries will also remain the same. When further queries are issued, the mobile device checks its cache to determine whether or not it can answer the query locally. In this way, some queries will be answered locally, response time is short, and energy consumption is relatively low. Hence, the problem of designing mobile devices is reduced to finding a way to generate a high cache hit rate without consuming too much resources in terms of computation power and storage capacity.

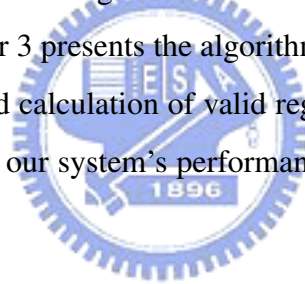
## 1.4 Contributions

The contributions of our thesis may be summarized as follows:

1. We propose a method that avoids the slow start of previous methods. Thus, our method will be superior to existing methods at the very start.
2. We propose a suboptimal way of merging circle valid regions and polygon valid region in the cache. This allows faster convergence than was previously possible.
3. We also propose methods that are well suited to k-NN query situations.

## 1.5 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we first describe the system architecture. Then, we present an overview of the existing works related to our thesis. We also classify these works according to their attributes. Chapter 3 presents the algorithms for combination of valid regions, cache replacement policy on the proxy and calculation of valid region for client. Chapter 4 describes extensions to k-NN queries. We evaluate our system's performance in Chapter 5. Chapter 6 concludes this thesis.



# Chapter 2

## Related Work

### 2.1 System Architecture

Figure 2.1 shows the mobile service system architecture which can provide NN and k-NN query services. We divide the mobile service system into service providers and service users. Service providers consist of base stations and database servers. Database servers answer NN and k-NN queries delivered from users. Base stations transmit users' queries to database servers, and also transmit answers to queries from servers to users. The base stations and data base servers are connected by fixed lines.

Mobile users issue queries which are transmitted from their mobile devices to database servers via wireless channels in the air. The answers to their queries are delivered in the reverse order.

### 2.2 Query Processing

When a mobile user makes a query, the mobile device searches its cache to determine whether the query can be answered locally. If the query location is the same as the one stored in its cache, the device will give the user a direct answer based on the data value of this cached location. Otherwise, this query will be transmitted from mobile device to database servers via wireless channels in the air. Then mobile user get his answers in the reverse way. After receiving answers, mobile device will update its cache if there is still free space in its cache. Otherwise, the device will take steps to replace the cache.

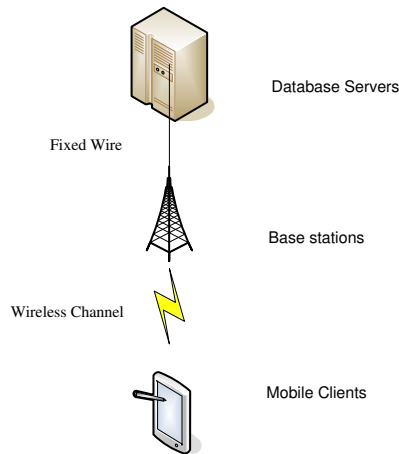


Figure 2.1: System architecture

## 2.3 Discussions

design issue (1) responsiveness timeliness server latency (2) energy constrain power management energy consumption For mobile clients, there are three critical issues which should be considered in order to design a good mobile service system: response time, energy consumption and cache replacement. Response time is the time from when the mobile client issues a query to the time when s/he receives an answer to this query from the database servers. Mobile clients need shorter response times in the mobile service system, as any delay in the query response may mean the answer given to the client is already obsolete. Mobile devices are powered by batteries, so each query must not consume too much energy. The sort of behavior with the highest energy consumption for mobile devices is communication with database servers. Mobile devices may be fitted with a cache which stores the query answer and its location. If the user's query location is the same as any of the cached locations, the answer to the query will also be the same as before. When we issue another query, the mobile device checks its cache to determine whether it can answer this query locally. In this way, some queries will be answered locally, response time is short, and energy consumption is low. The problem of cache replacement stems from the fact that mobile devices are smaller than database servers, so we have to find a way to obtain a high cache hit rate without taking up too much storage resources Servers receive queries from mobile users and calculate answers to those queries. Servers relay answers to mobile users via wireless channels. From the view point of servers, the servers hope to serve more mobile users in the finite resource of calculation and storage. Another important issue is how efficiently servers use limited wireless chan-

nels. In addition to LDQ answers, the servers can send additional information to mobile users, e.g. the valid region. Mobile users can use this additional information as a basis to process some future queries locally.

## 2.4 Voronoi Diagrams and Valid Regions

Using valid regions is a good way of resolving questions concerning low link quality, low power use and user mobility. We will discuss valid regions from two different view points. For identical users, [13] says that the answers may be invalid for moving users as the servers are too slow to respond to questions in time. If we can use the valid region, and if users are still in the valid region, users can directly answer these questions by themselves. Only when users move out of the valid region do they need to reissue questions. The second problem is that if the server responds slowly and user changes position, then the solution may be invalid. If the user uses valid regions, it can still be determined whether or not the answer is valid by referring to the valid region. Otherwise, it is necessary to reissue the query to the server. Different users may also benefit by using valid regions, as some places have "hot" queries such that when the system caches valid regions for these queries, users can get answers directly from the cache. For example, a train station is a "hot" place where many people issue queries about the nearest hotels or restaurants. The answers to these queries are often the same, so if the system caches them, other users can access the cache directly without requiring the server to recalculate answers. In this way, response time is faster and system performance is better. There are three ways to provide valid region data. One is to use a pre-calculated Voronoi diagram, the second is to use a valid region online, and the last is to use a proxy to learn valid regions. We will discuss these ways in more detail in the following sections. A Voronoi diagram is a set of polygons, called Voronoi cells, generated by data objects. A Voronoi cell is a collection of points whose nearest neighbor is the generator data object. Thus, Voronoi cells partition total space. Figure 2.2, below, is an example of a Voronoi diagram, where each hospital is an object of interest and is contained by a valid region. Servers must calculate the valid regions of the objects and store them, thus providing service for valid regions. When a server receives query, it maps it onto a valid region of some object, then relays the data for this object to the user. This is very fast, as servers must calculate the valid regions for any object only once and can then use them many times. But valid regions are often irregular polygons and storing their data takes up a lot of space,

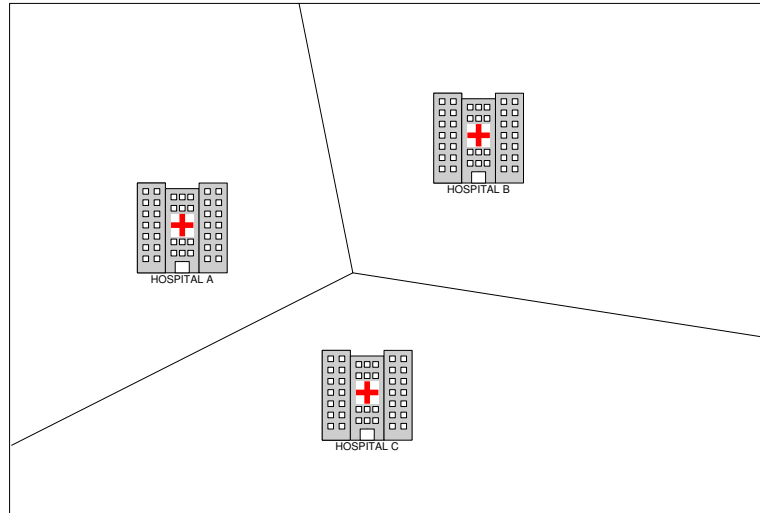


Figure 2.2: Voronoi diagram

so this method is only suitable for NN queries, and not for k-NN queries.

## 2.5 Prior Approaches

Most existing approaches to providing valid regions to users are based on servers using pre-computed Voronoi diagrams or computing valid regions online, or by proxies learning valid regions from history records. The first approach requires pre-computation of the Voronoi diagrams for data objects and stores them as index structure. During run time, the server answers a nearest neighbor query  $q$  as follows. First, the server finds the Voronoi cell which contains this query by exploiting the pre-stored index structure. Then, the server takes the generator data object for this Voronoi cell as the answer and provides all or a subset of this Voronoi cell as a valid region for the query. This approach has the advantage that the server constructs the Voronoi diagrams only once and can use them many times; in this way, users benefit from a fast response time. However, a disadvantage of this approach is that the server requires in advance a pre-determined value of  $k$  in order to pre-compute the corresponding  $k$ -order Voronoi diagrams; but this is impossible for real-world scenarios, since the value of  $k$  is determined during run time. An alternative method would be to pre-compute and store Voronoi diagrams for all possible values of  $k$ , but  $k$ -th order Voronoi cells usually have complex shapes and the corresponding storage overhead would be very high. Traditional approaches assume that the server relays the valid region to the user directly. But this is impractical in the real world as it consumes excessive calculation

and storage resources. For this reason, in the second approach the server provides the user only with query results, and not with a valid region.

The second approach uses the novel algorithms proposed by [14] to enable servers to compute the valid regions in real time. The general way this approach answers a  $k$ -nearest neighbor query  $q$  is as follows. First, the server acquires  $k$  data objects as a result of  $q$ . Then, it recursively issues TPkNN queries to find other data objects, called influence sets, which contribute the valid region for  $q$ . Finally, it transmits the query answer and influence set to the user. The user's mobile device can calculate the valid region by itself by using the influence set. The server thus does not need to set aside space to store Voronoi cells; and in addition, this approach can provide  $k$ -order Voronoi cells without restrictions. The drawback of applying this approach to provide valid regions is that the server performs poorly when queries are dense since it needs extra I/O resources in order to calculate valid regions for each query in real time. Consequently, users will suffer from long wait and slow response times, and this situation will deteriorate in instances of huge crowds or burst floods of queries. Moreover, the user's mobile device needs additional resources in order to calculate valid regions, and this situation is aggravated when  $k$  is large. Consider now a third possible approach which posits a proxy between server and users. The proxy learns to estimate valid regions using history records referring to the server and users, and provides these learned valid regions to users under certain conditions. Figure 2.3 (a) shows the history records in the proxy cache. Figure 2.3 (b) shows, the actual valid regions (AVR), a Voronoi diagram and the estimated valid regions (EVR).

But there are still some drawbacks to this approach. First, it is only appropriate for nearest neighbor queries and does not work properly for  $k$ -nearest neighbor queries. Second, it is not efficient enough. In terms of efficiency, the existing proxy method has two main drawbacks. One is the slow start phenomenon and the other is overly rapid EVR combination. To contribute a valid region for an object, the proxy needs at least three different queries whose solutions are this object. This means that most queries will suffer a cache miss at first: this phenomenon is called the slow start. Next, the proxy faces the other problem of overly rapid EVR combination. The existing approach assumes that only the query point and EVR polygon are merged. But if the proxy receives not only a query point but also a circle valid region, this method does not make good use of all the information. Figure 2.4 shows the current and optimal ways of merging EVR. We compare these three kinds of methods in Table 2.1.

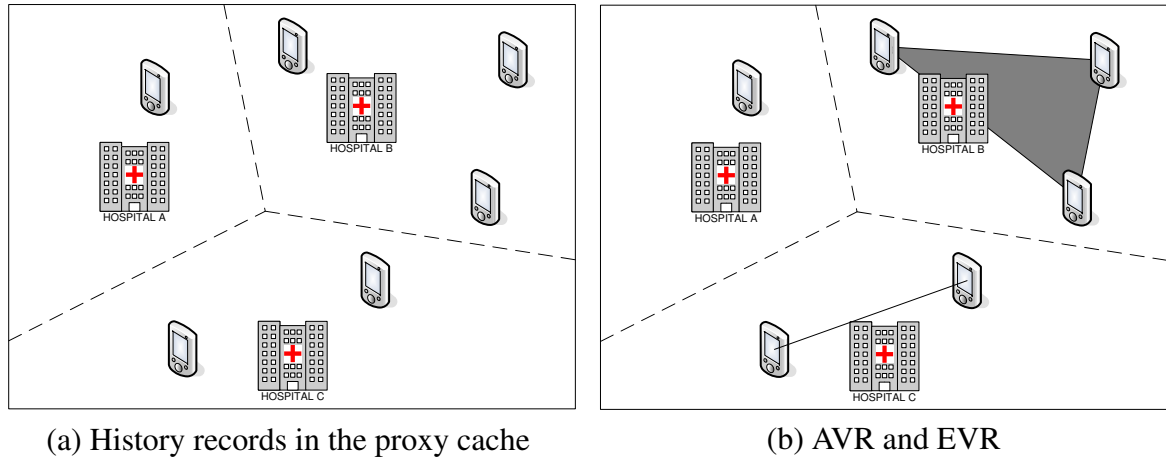


Figure 2.3: Learning by history

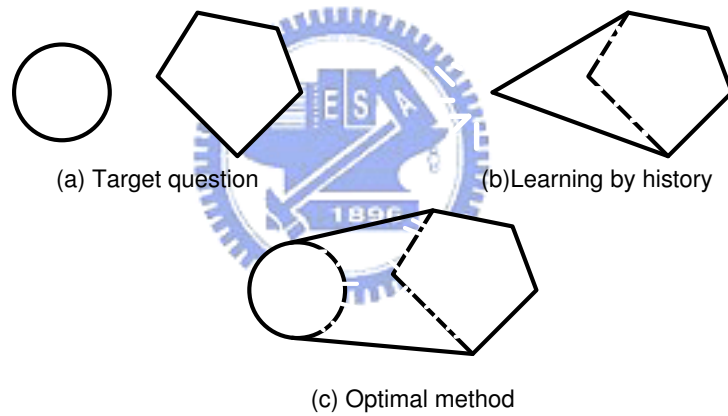


Figure 2.4: EVR combination

Method	Server speed	Space	Response time	$k$ NN query
Pre-calculation	Fast	Large	Medium	None
Online calculation	Slow	Small	Medium	Support
Learning by history	Medium	Medium	Fast	None

Table 2.1: Cartogram



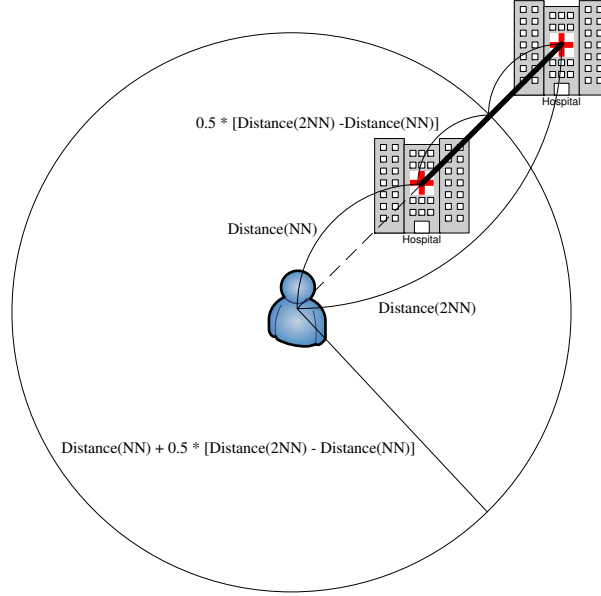


Figure 2.5: Valid region of 2-NN query

## 2.6 k-Nearest Neighbor Search for Moving Query Points

[10] proposes an idea that can be used for any value of  $k$ . When a server receives a  $k$  nearest neighbor query, it evaluates the query and gives  $(k+1)$  nearest neighbors to the client. Assume that  $Distance(k)$  and  $Distance(k+1)$  are the distances of the  $k$ -th and  $(k+1)$ -th nearest neighbors from the query issuer, respectively. Then the circle whose center is the query point and whose radius  $r$  is half of  $Distance(k+1) - Distance(k)$  is the valid region of  $k$  nearest neighbors of  $q$ . That is

$$r = \frac{Distance(k+1) - Distance(k)}{2}$$

Figure 2.5 shows an example for a nearest neighbor query at position  $q$ .

## 2.7 Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments

[13] proposes two ways of evaluating the values of valid regions in the cache. One is known as PA and the other PAID. When the cache is exhausted, the server or proxy selects the valid region with the lowest value as the "victim". The PA method considers the access rate and the area of objects:

object with higher access rates or larger areas have a larger cache value. PAID considers the distance between valid regions in the cache and query point more than PA. The shorter the distance, the higher the probability that the valid region will be reused by clients. Apart from this elimination of "victims", [13] proposes two further methods for reducing the storage cost of valid regions. First, valid regions can be degenerated from polygons to inscribed circles. If each point of a polygon costs two bytes and a polygon has  $n$  points, it will cost  $2n$  units in storage. But an inscribed circle costs only 3 units (the center costs 2 units, and radius costs 1 unit). The drawback of this method is that in some cases it is insufficiently accurate. A second method reduces points in the valid region, usually by selecting the points which contributes less to the area than others. The benefit of this approach is its increased accuracy; but the cost of evaluation is higher.



# Chapter 3

## Query Processing for NN Queries

### 3.1 The System Model

In this section we describe the applied architecture. We assume that the coordinate system is a Cartesian coordinate system, i.e., each position is represented by two pairs of numbers. The system includes three components: database servers, proxy, base stations. Mobile clients communicate with the server via the base station and wireless links. Base stations, proxy servers and database servers intercommunicate using fixed links. A mobile client can move randomly around the map (with restrictions on speed and direction) without losing the connection. The client can also identify its position using the Global Positioning System (GPS), and issue queries whose answers depend on its location. Before sending such requests to servers, the client first checks its local caches. The database servers and proxy servers provide nearest neighbor or k-nearest neighbor answers to mobile clients. We assume that database servers do not provide any valid region information with answers, but proxy servers will provide valid regions using learned histories. We also assume that valid regions are organized in geometric polygons and stored in databases in the form of link lists of points. We separate the design of the proxy into two parts: one part determines whether the query cache has been hit, and the other updates the proxy cache. The first part is responsible for receiving the query and then, on the basis of the location of the query and the information in the cache, deciding whether the cache has been hit or missed. If the cache has been missed, the proxy changes the NN query into a 2NN query and relays it to the server. When the proxy receives the 2NN query solutions, the second part translates the 2NN solutions into NN solutions and valid regions for NN queries. In addition, the proxy updates the cache with new NN valid regions.

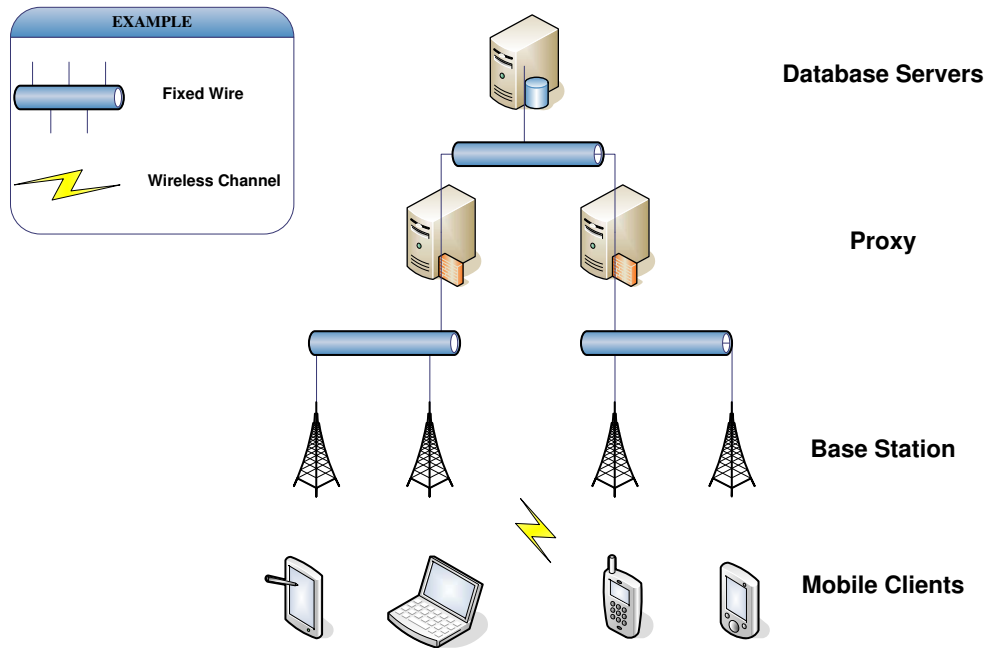


Figure 3.1: The system model

The performance of first part depends on how fast it can detect cache hits/misses, and how fast it can calculate the EVR for the client, because cache detection and EVR calculation are the overheads from the original system. The performance of the second part depends on whether it uses storage efficiently, because storage is also an overhead from the original.

## 3.2 Overview

Figure 3.2 shows the overview of our method:

1. The 1NN query,  $(1, x_c, y_c)$ , will be transmitted to "Cache Lookup" of proxy from client at the position  $(x_c, y_c)$ .
2. The "Cache Lookup" uses Rtree procedure to determine whether this query is cache hit or not.  
If cache hit, this query will be passed to "EVR for client".  
If cache miss, this query will be sent to "Encase NN to 2NN Query". Then, "Encase NN to 2NN Query" changes original NN query into 2NN query,  $(2, x_c, y_c)$ , and request server this new query.
3. After receiving solutions,  $(data1, x1, y1)$  and  $(data2, x2, y2)$ , from server, "Decode 2NN to

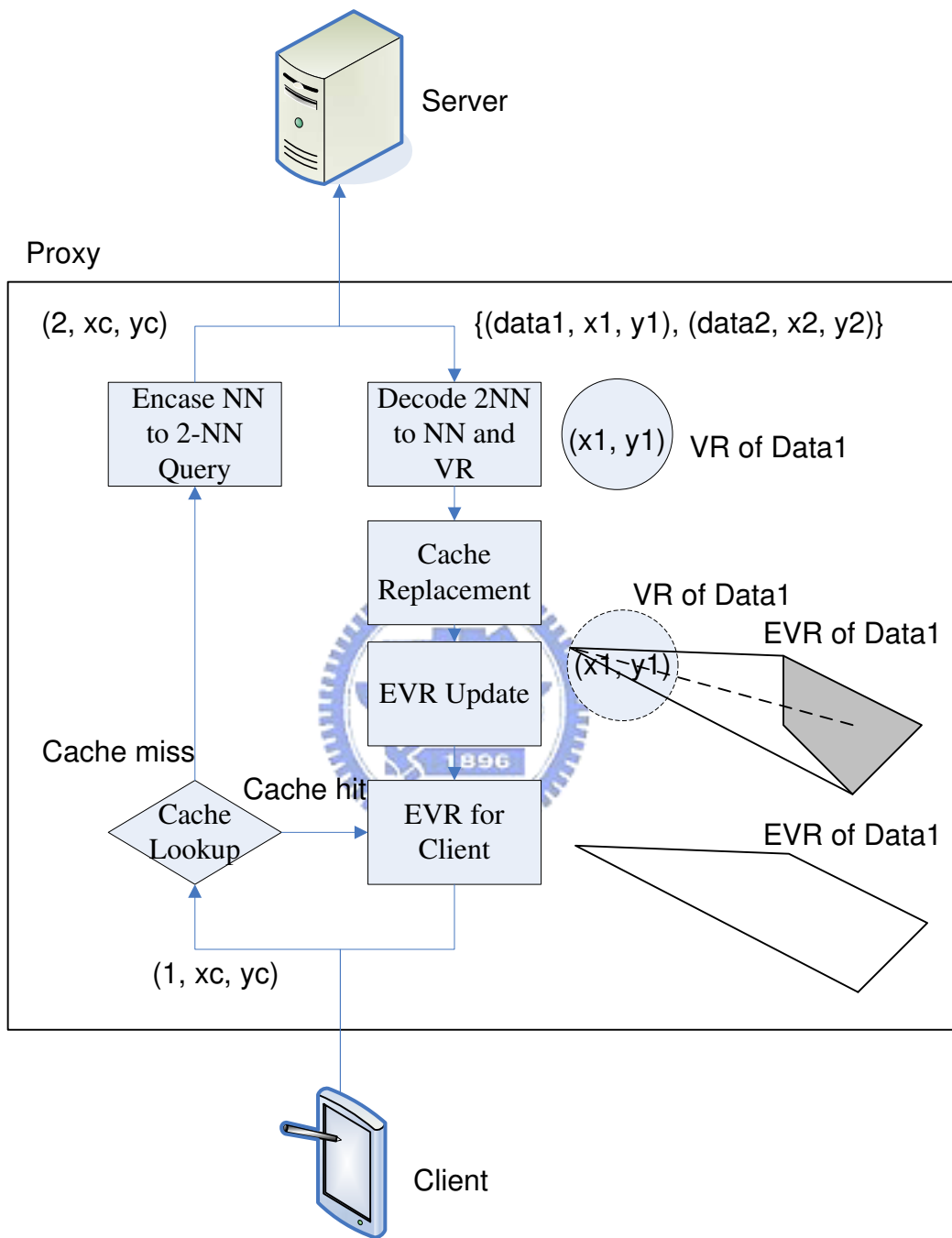


Figure 3.2: Flowchart

NN and VR” decodes these solutions into answers of original query, (data1, x1, y1), and its corresponding EVR, ”VR of Data1”,.

4. ”Cache Replacement” will use Cache-Replacement algorithm to reduce some polygons when there is not enough space to update it cache.
5. ”EVR Update” will deal with update of ”VR of Data1” and ”EVR of data1” in the proxy cache.
6. ”EVR for client” will use EVR-for-Client algorithm to reduce n-polygon to 4-polygon. Then, the answers and its corresponding modified EVR will be returned to client.

### 3.3 Organization of the Proxy Cache

Traditionally, Voronoi diagrams are structured with D-trees or grid cells. Each polygon is represented in the form of a series of points or edges, and there is a need for methods which can search the polygons faster. But the EVRs in our cache are not the same as Voronoi diagrams; therefore, we do not use the traditional method to organize our EVRs [4] [3] [11] [7]. Instead, we converge each polygon with an minimum bounding rectangle (abbreviated as MBR), then organize these MBRs with R-trees or its variations. The R-tree method has been proven to work well in this situation.

### 3.4 Cache Lookup

We divide the component for cache hit detection into two parts. One part filters impossible cases, while the second part determines whether the possible cases are hits or misses. In the case of a miss, the query is transferred to the server; otherwise, it is transferred to the EVR calculator. We enclose the EVR with an MBR, then organize them with R-trees. First we determine which MBR contains the query: if the query has come from someone MBR, this means that the query may also fall in the EVR. Once it is known that the query belongs to some MBR of some object, then it must be determined whether the query falls in the EVR. Therefore, we construct a segment between the location of the query and the location of the object. Determining whether the query

falls in the EVR is to the same as detecting whether the segment crosses any edge of any polygon in the EVR.

### 3.5 Initial EVR Update

Because we require at least three different points to form an EVR polygon for some object, the first three queries whose solutions are the same object may be read as cache misses by existing methods. This is called the phenomenon of the slow start and takes place at the beginning of proxy service. In order to eliminate this phenomenon, we have designed an algorithm that can learn EVR polygons after the first cache miss for some object. In our design system, the cache-missed NN query is converted to a 2-NN query and sent to the database server for computation. The proxy then splits the solutions for the 2-NN query sent from the database server into an answer to the original NN query and the corresponding valid region. This information is then used to update the proxy cache and relayed to the mobile client through the base station. When we update the EVR of some object for the first time, we apply a Fast-Start algorithm to construct the EVR of that object. The input of this algorithm is the query point together with its valid region and object point. In our Fast-Start algorithm, we search for an inscribed  $n$ -regular polygon in the valid region of the query point and merge this polygon with an object point to form a new EVR for the object. The inscribed  $n$ -regular polygon can be found using De Moivre's Theorem or a rotation matrix. This Fast-Start algorithm will use incremental right-hand algorithm to make a new convex hull, and we describe it in later subsection. Figure 3.3 shows these cases.

**Lemma 1.** *If an  $n$ -vertex polygon is  $n$ -regular, it will have the largest area within a circumscribed unit circle.*

*Proof.* Let  $A$  be the area of the  $n$ -polygon within a circumscribed unit circle, and  $\theta_n$  is the corresponding central angle of edge  $e_n$  in this polygon. Then,

$$A = \sum_{\forall n} \frac{1}{2} \sin \theta_n$$

To maximize  $A$  is the same as to maximize

$$\sum_{\forall n} \sin \theta_n$$

or, equivalently, to maximize

$$\sum_{\forall n} \sin^2 \theta_n$$

By Cauchy Schwarz Inequality,

$$\left(\sum_{\forall n} \sin \theta_n\right)^2 \leq n \sum_{\forall n} (\sin \theta_n)^2$$

Equality holds if and only if

$$\sin \theta_1 = \sin \theta_2 = \dots = \sin \theta_n$$

This means that regular n-polygons has the largest area within a circumscribed unit circle.

□

**Definition 1.** We define the cost value as  $A/C$ , where  $A$  is the total area and  $C$  is the vertices of this area.

**Lemma 2.** Regular polygon will have the maximum cost value when  $n$  is 4.

*Proof.* By definition of cost value, we know that

$$A/C = \frac{\frac{1}{2}n \sin\left(\frac{2\pi}{n}\right)}{n}$$

that is,

$$A/C = \frac{1}{2} \sin\left(\frac{2\pi}{n}\right)$$

, when



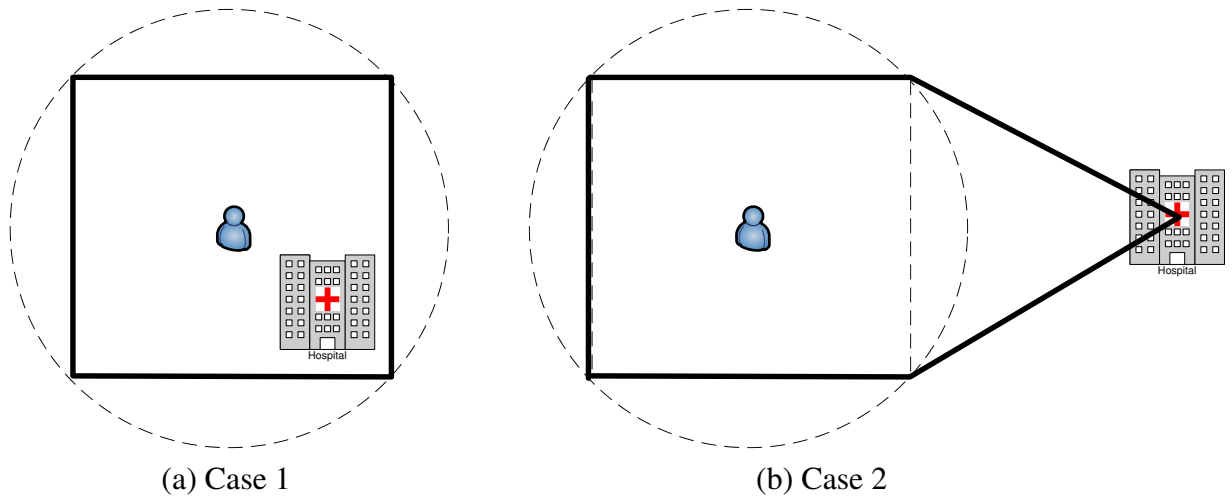
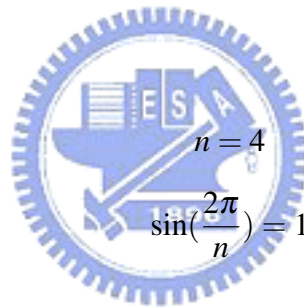


Figure 3.3: Fast start

$$\sin\left(\frac{2\pi}{n}\right) = 1$$

has maximum cost value.

We also know that when



□

**Algorithm Fast-Start**

- 1: Find an inscribed 4-regular polygon of the valid region of this query point
- 2: **if** (The distance between the query point and object point is larger than the radius of the valid region of the query point) **then**
- 3:     Combine this polygon with an object point to make a new polygon using the Incremental Right-Hand Algorithm
- 4: **end if**
- 5: Let this polygon be the the EVR of object
- 6: **return** EVR of object

### 3.5.1 Incremental Right-Hand Algorithm

Our method of combining EVR polygons frequently uses a convex-hull algorithm. For this reason, we propose a fast incremental convex-hull algorithm (ICHA) that merges an existing convex hull of a set of  $n$  points with a new point outside the section. The newly formed polygon is a  $p$ -vertex convex hull, where  $p \leq (n + 1)$ . This algorithm runs in  $O(n)$  time. The input of the algorithm is an  $n$ -vertex convex hull which contains at least three points and a target point  $p_q$  outside the polygon. First, it finds a center point  $p_c$  inside the input polygon; then, it calls the functions  $\text{Angle}(p_q, p_c, p_r)$  which return the angle formed by the lines  $\overline{p_q p_c}$  and  $\overline{p_c p_r}$ . Instead of actual angle, the function  $\text{Angle}(p_q, p_c, p_r)$  will return the value  $\frac{|\overline{p_q p_r}|}{\overline{p_q p_c} \cdot \overline{p_q p_r}}$ . Then, the ICHA compares these angles to find the points which form the largest angles on the two sides of  $\overline{p_q p_c}$ . Finally, it connects the points found on the boundary to the target point and deletes all the points inside the newly formed polygon. The outcome of this algorithm is a  $p$ -vertex convex polygon that contains the target point on its boundary, where  $p \leq (n + 1)$ . The following are some of the properties used in the ICHA. And this algorithm will be used in EVR-Update algorithm later.

**Lemma 3.** *The angle between  $\overline{p_q p_c}$  and  $\overline{p_q p_r}$  and the value of  $\frac{|\overline{p_q p_r}|}{\overline{p_q p_c} \cdot \overline{p_q p_r}}$  have positive correlation.*

*Proof.* Let  $\theta$  be the angle between  $\overline{p_q p_c}$  and  $\overline{p_q p_r}$ .

$$\cos \theta = \frac{\overline{p_q p_c} \cdot \overline{p_q p_r}}{|\overline{p_q p_r}| |\overline{p_q p_c}|}$$

that is,

$$\theta = \arccos \frac{\overline{p_q p_c} \cdot \overline{p_q p_r}}{|\overline{p_q p_r}| |\overline{p_q p_c}|}$$

We know that arccos function is strictly decreasing in  $[0, \pi]$ , and this means that  $\theta$  and  $\frac{|\overline{p_q p_r}|}{\overline{p_q p_c} \cdot \overline{p_q p_r}}$  have positive correlation. □

**Definition 2.** *To find the maximal angle is the same as to find the maximal value of  $\frac{|\overline{p_q p_r}|}{\overline{p_q p_c} \cdot \overline{p_q p_r}}$ .*

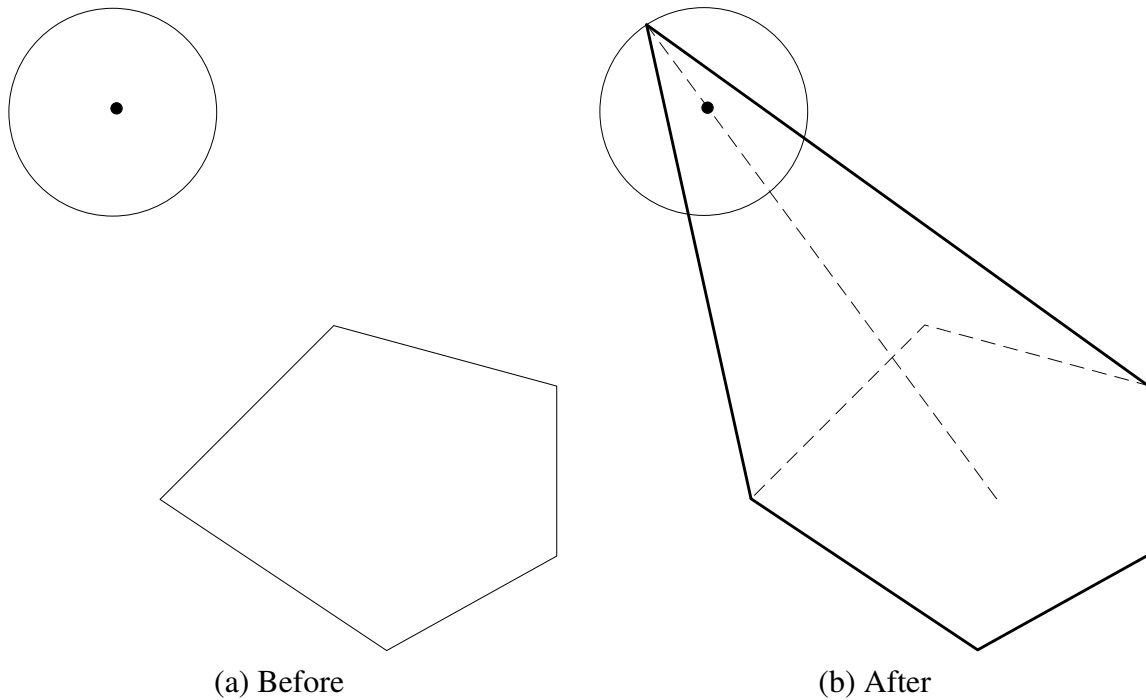


Figure 3.4: EVR update

By definition, we can see that we can find the target points at most  $n$  times and the time complexity of ICHA is  $O(n)$ .

**Algorithm Incremental-Right-Hand-Algorithm**

- 1: Find a point  $p_c$  inside the input convex hull
- 2: Form a vector  $\overrightarrow{p_q p_c}$
- 3: **for**  $i \leftarrow 1$  to  $n$  **do**
- 4:    $a \leftarrow \text{ANGLE}(\overrightarrow{p_q p_i}, \overrightarrow{p_q p_c})$
- 5:   Find the vertex  $s, t$  that have maximal angle on two side of  $\overrightarrow{p_q p_c}$
- 6: **end for**
- 7: Connect  $s$  and  $p_s$
- 8: Connect  $p_t$  and  $s$
- 9: Delete all points inside the new formed polygon
- 10: **return** Convex hull

**Procedure Incremental-Right-Hand-Algorithm**

- 1: **return**  $\frac{|\overrightarrow{p_q p_r}|}{\overrightarrow{p_q p_c} \cdot \overrightarrow{p_q p_r}}$

### 3.6 EVR Update

The proxy thus have valid regions for some objects for some time, and these valid regions should be polygons. Each object, after a query, has a polygonal valid region and a new circle valid region. We have to merge a polygon and a circle. Figure 3.4 shows these cases. In this algorithm, we first form a vector  $u$  from an object point and a query point. Then we have the direction vector and norm  $r$  of this vector, where  $r$  is the EVR radius of the query point. From this results a new point which can be merged with the polygon from the new vector and object point.

Finally, we apply the right-hand algorithm to merge the point and the polygon. The algorithmic form of the algorithm is as follows.

#### Algorithm EVR-Update

- 1:  $\Delta y \leftarrow c_y - q_y$
- 2:  $\Delta x \leftarrow c_x - q_x$
- 3:  $\theta \leftarrow \tan^{-1} \frac{\Delta y}{\Delta x}$
- 4:  $y' \leftarrow q_y + r \sin \theta$
- 5:  $x' \leftarrow q_x + r \cos \theta$
- 6: Combine  $(x', y')$  with original EVR to form a new polygon using the Incremental Right-Hand Algorithm
- 7: Let this polygon be the new EVR of the object
- 8: **return** EVR of object



### 3.7 Cache Replacement

This section will do two things. One is to determine which valid region should be the "victim". The other is to decide how to shrink the valid region in order to free up space. When we execute the cache replacement policy, the system usually runs for a while, and it can be assumed that the valid region with the smallest area is the least popular. We can also calculate the usage of each valid region, and the valid region with the smallest usage is also a candidate to be the victim.

Once the victim has been selected, we need to find a way to reduce its valid region in order to release space. By lemma we know that the polygon with 4 vertices will have the largest cost value, and we assume that larger areas have a larger number of cache hits. But, it is very difficult and time consuming to find the quadrangle with the largest area in target  $n$ -polygon. An trivial

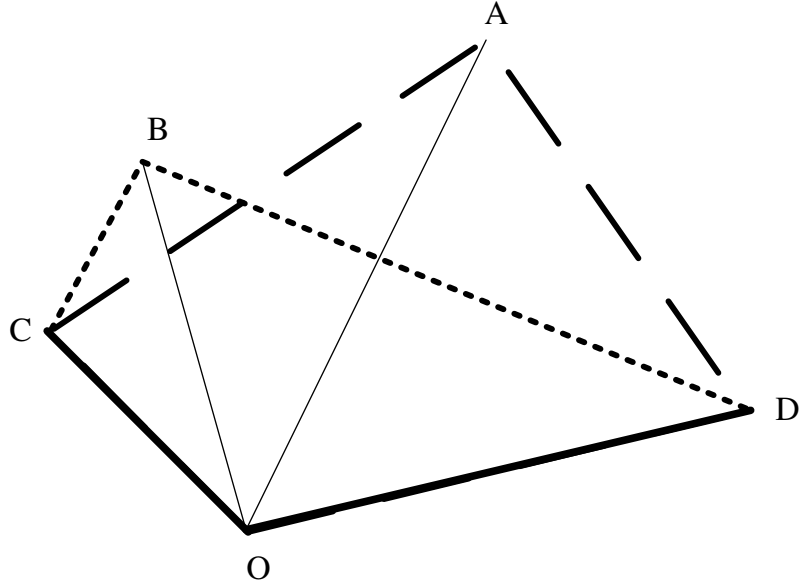


Figure 3.5: Example of lemma

way is that we random select four vertices in the target polygon to form an quadrangle. Another greedy way is that we remove the point which contributes the least area to target valid region until it is 4-polygon. Here we propose an fast heuristic way to select an quadrangle that will has larger area in original valid region. We set the heart of this polygon as original and divide vertices into four quadrants. We choose 1, 3 or 2, 4 quadrants that are not all empty to select vertices which have larger distances between origin. Then we select another two vertices which also have larger distances between origin in the other two quadrants. By lemma we can known that this way will make larger area. Figure 3.6 shows the example of this method.

**Lemma 4.** *Let  $OA$  and  $OB$  between angle  $COD$ , and  $|OA| \geq |OB|$ . The area  $OCAD$  will bigger than area  $OCBD$ . Figure 3.5.*

*Proof.* Let  $\theta_{ij}$  be the angle between edges  $O_i$  and  $O_j$ . The area of  $OCAD$  is the sum of triangles  $OCA$  and  $OAD$ .

That is,

$$area_{OCAD} = \frac{1}{2}|OC| \times |OA| \times \sin \theta_{CA} + \frac{1}{2}|OA| \times |OD| \times \sin \theta_{AD}$$

, equivalently,

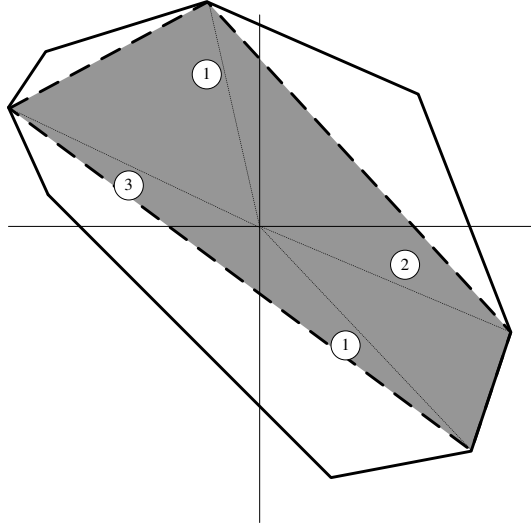


Figure 3.6: Example of cache replacement

$$areaOCAD = \frac{1}{2} \sqrt{(|OC| \times |OA|)^2 + (|OA| \times |OD|)^2} \times \sin(\theta_{CA} + \theta_{AD})$$

For the same reason,

$$areaOCBD = \frac{1}{2} |OC| \times |OB| \times \sin \theta_{CB} + \frac{1}{2} |OB| \times |OD| \times \sin \theta_{BD}$$

or,

$$areaOCBD = \frac{1}{2} \sqrt{(|OC| \times |OB|)^2 + (|OB| \times |OD|)^2} \times \sin(\theta_{CB} + \theta_{BD})$$

We know that

$$\theta_{CA} + \theta_{AD} = \theta_{CB} + \theta_{BD}$$

and

$$|OA| \geq |OB|$$

Thus,

$$areaOCAD \geq areaOCBD$$

□

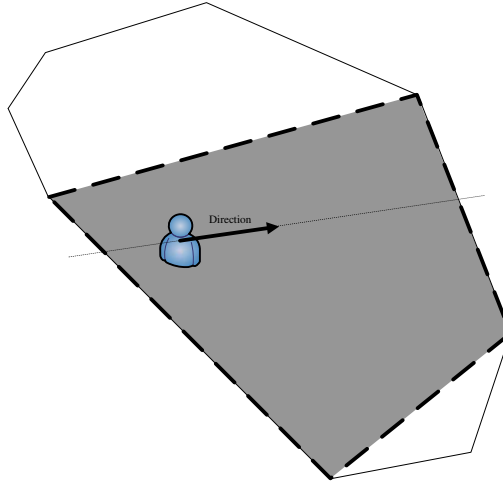


Figure 3.7: Example of client's EVR

**Algorithm Cache-Replacement**

- 1: Set heart of this polygon as origin
- 2: Select 1, 3 or 2, 4 quadrants that all have vertices
- 3: Find vertices a, b that have longest distances from origin in these two selected quadrants
- 4: Find other two vertices c, d that have longest distances from origin between a and b
- 5: **return** Quadrant with a, b, c and d as vertices

**3.8 EVR for Client**



Considering the parameters of calculating ability, small storage and power consumption, we provide an EVR from proxy to client in the form of a quadrant. We will use the information of client's movement to provide the benefit of predicting. Figure 3.7 shows the example of this method.

**Algorithm EVR-for-Client**

- 1: Form a line l whose slope is the same as client's position and its predicted point.
- 2: Find the two edges which intersect line l
- 3: **return** Quadrant these two edges

# Chapter 4

## Extensions to k-NN Queries

The k-NN situation is more complex Figure exhibits all the cases in the k-NN query. The value of EVR  $k$  is the valid region cached in the proxy, and the query  $k$  is the query which interests the client. We can classify them as cache hits or misses, as follows.

1. Total cache miss: There is no valid region in the proxy corresponding to the received k-NN query. In the figure, the client issues a 5-NN query, but this query point does not belong to any valid region cached in the proxy. Figure 4.1 (a) shows this case. In total cache miss case, we will do things the same as cache miss case in NN situation.
2. Total cache hit: The k-NN query point belongs to some valid region and the value of  $k$  is the same as that of the valid region. In the figure, the client issues a 5-NN query and this query point belongs to a valid region whose  $k$  value is 5. The answers in this valid region are the results of the query. Figure 4.1 (b) shows this case. In total cache hit case, we will do things the same as cache hit case in NN situation.
3. Partial cache hit, case 1: The number  $k$  of the query is smaller than the number  $k$  of the cached valid region. The query point belongs to some cached valid region whose  $k$  value is larger than the  $k$  value of the query. In the figure, the client issues a 3-NN query and this query point belongs to a valid region whose  $k$  value is 5. In this case, the partial data items for the valid region are the results of the query. The proxy must sort all the data items to get 3 NN solutions for the client. Figure 4.1 (d) shows this case. The proxy will sort all



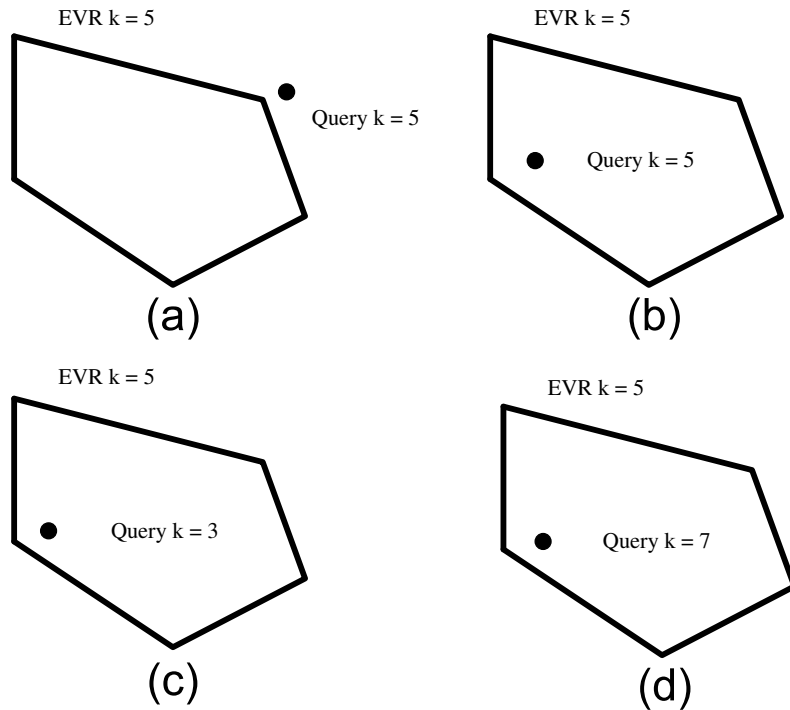


Figure 4.1: k-NN queries

solutions in the cache hit EVR to find the answers. Then, it will reply these answers and this EVR to client.

4. Partial cache hit, case 2: The number  $k$  of the query is larger than the number  $k$  of the cached valid region. The query point belongs to some cached valid region whose  $k$  value is smaller than the  $k$  value of the query. In the figure, the client issues a 7-NN query and this query point belongs to a valid region whose  $k$  value is 5. Figure 4.1 (c) shows this case. Here, the proxy returns the partial answers in the proxy to client and transfer this cache miss query to server. After receiving solutions, the proxy replies the other answers and EVR to client.

We describe these ideas in the following algorithm.

**AlgorithmKNN-Cache-Lookup**

- 1:  $S = \text{Procedure-RTEE}(q)$  /\* Procedure-RTEE( $q$ ) will return the EVR  $E$  that contains  $q$ ,  $S$  is the set of  $E$  \*/
- 2: **if**  $|S|$  not empty **then**
- 3:     **if** There exists  $E$  whose  $E_k$  equals  $q_k$  **then**
- 4:          $E' = \text{Algorithm-Client's-EVR}(E)$
- 5:         Send  $E'$  and  $E_{solution}$  to client /\*  $E_{solution}$  is the data item solutions in  $E$  \*/

```

6:  else
7:    Devide S into  $S'$  and  $S''$  by  $E_k$  /* We assume the E in  $S'$  will have  $E_k$  larger than k, and
    E in  $S''$  will have  $E_k$  smaller than k */
8:    if  $|S'|$  not empty then
9:      Find the E' which has minimal  $E_k$  in  $S'$ 
10:     Sort  $E'_{solution}$  and pick up  $q_k$  numbers as solution  $E_{q_{solution}}$ 
11:     Eq = Algorithm-Client's-EVR(E')
12:     Send Eq and  $E_{q_{solution}}$  to client
13:   else
14:     Find the E'' which has maximul  $E_k$  in  $S''$ 
15:     Let  $E''_{solution}$  as partial solutions  $E_{q_{partialsolution}}$ 
16:     Resend this query to server
17:     Send  $E_{q_{partialsolution}}$  to client /* After receiving answers from server, the proxy will
    sends  $E_{q_{solution}}$  and Eq to client */
18:   end if
19: end if
20: else
21:   Resend this query to server /* After receiving answers from server, the proxy will sends
     $E_{q_{solution}}$  and Eq to client */
22: end if

```



# Chapter 5

## Simulation

We use JSIM [15] and Spatial Index Library [16] to build our simulator. The simulator is used to evaluate the performance of our methods when applied. In the simulation, it is assumed that the servers only provide answers to clients without valid regions. The results of the simulation show that our method is superior to the existing method in terms of response time and database workload. Moreover, it also shows that our method also works well in k-NN environments.

### 5.1 Simulation Model

The environment of our simulation is similar to those in [5] [13]. (The system parameters are listed in Table 5.1.) We model the simulation area in a "wrapped-around" rectangular area whose size is  $spaceX * spaceY$ . The area contains  $objectNum$  objects and  $clientNum$  clients. It is trivial that the number of clients and the number of objects in the system are the same. Each mobile client can issue NN and k-NN queries. When the client receives the answers to the issued query, it will wait for some time,  $queryInterval$ . We assume there is some local storage site whose size is  $clientCacheSize$  for each client. Before a client issues a new query, it will first check its cache. In addition, each client moves in a "random walk" fashion. The speed and direction of the client are limited by the values of  $speedInterval$  and  $directionInterval$ , respectively. Let  $DatabaseServiceTime$  be defined by the time in which the database executes each query. Let  $ProxyServiceTime$  be the time in which the proxy determines whether or not each received query

Parameters	Default Value
spaceX	1000 m
spaceY	1000 m
objectNum	100
clientNum	100
queryInterval	10 s
maxSpeed	1 m/s
maxDirection	180 degree
dataBaseServiceTime	0.14 s
proxyServiceTime	0.0001 s

Table 5.1: Simulation parameters

is a cache hit. The size of the proxy storage is limited by the *proxyCacheSize*.

## 5.2 Simulation Results

### 5.2.1 Effect of Fast Start

Here we will discuss the effect of fast start at the first in four dimensions. We observe the simulation results of area ratio of valid regions in proxy and AVR, proxy hit rate, waiting time of each query, and average edges stored in the proxy.

We first discuss the area ratio of valid regions in proxy and AVR. Larger area ratio means much valid region area learned by proxy. Figure 5.1 shows that we can get much EVR than [5] at very start. By this reason, Figure 5.2 shows that the proxy hit rate of our method is superior to [5]. The client in the simulation with fast-start has a lower waiting time. This is because our method organizes larger EVRs and has larger cache hit rate at the start and fewer clients fall victim to cache misses via the proxy. Figure 5.3 shows these results. However, the cost of our method is almost the same as [5], Figure 5.4. That means we succeed in overcoming the slow start of [5].

### 5.2.2 Effect of Proposed EVR Update

Here we will discuss the effect of proposed EVR Update in four dimensions. We also observe the simulation results of area ratio of valid regions in proxy and AVR, proxy hit rate, waiting time of

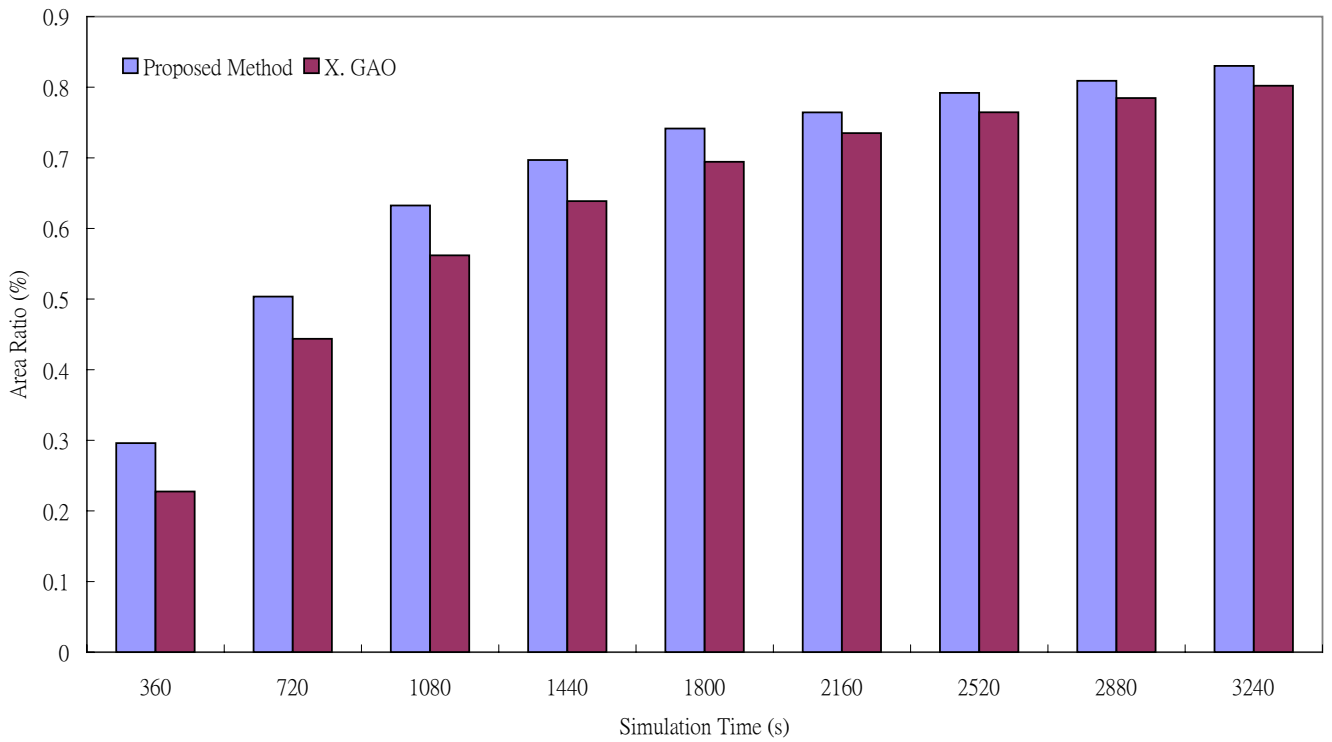


Figure 5.1: Area ratio for NN query at very start

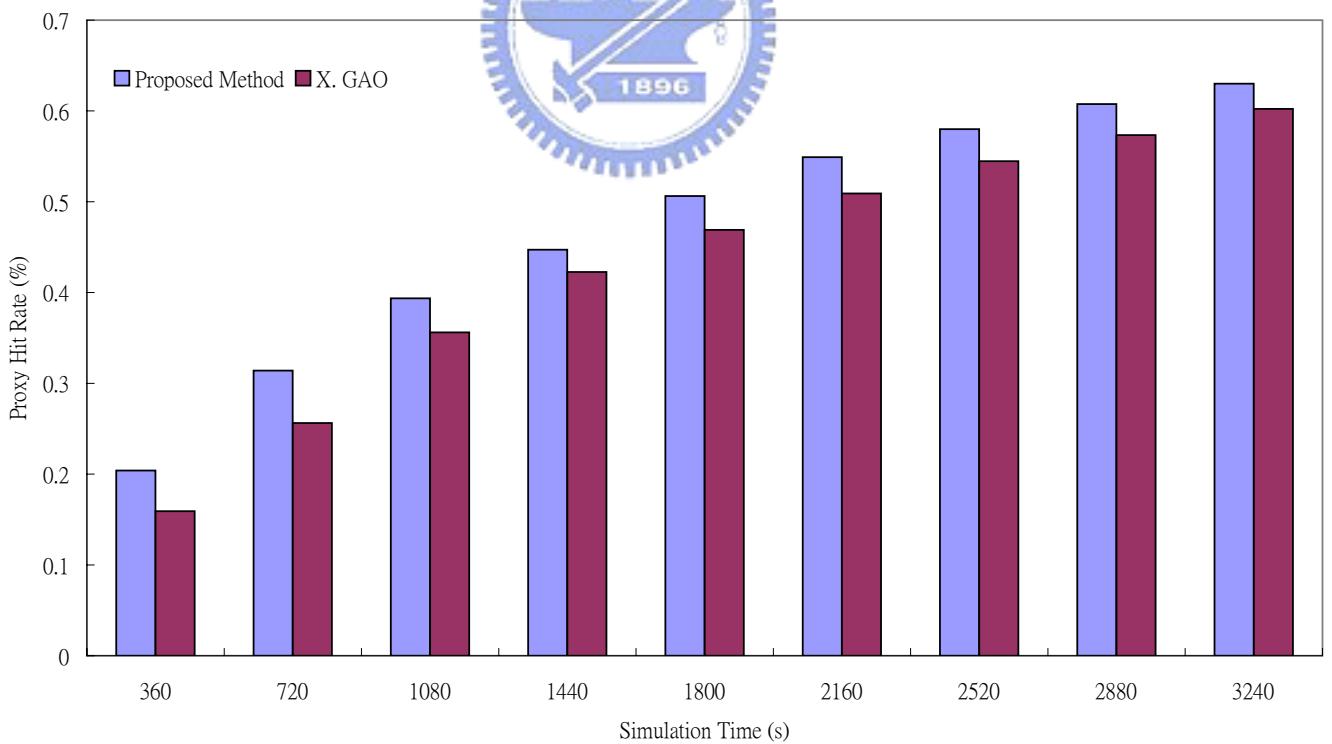


Figure 5.2: Proxy hit rate for NN query at very start

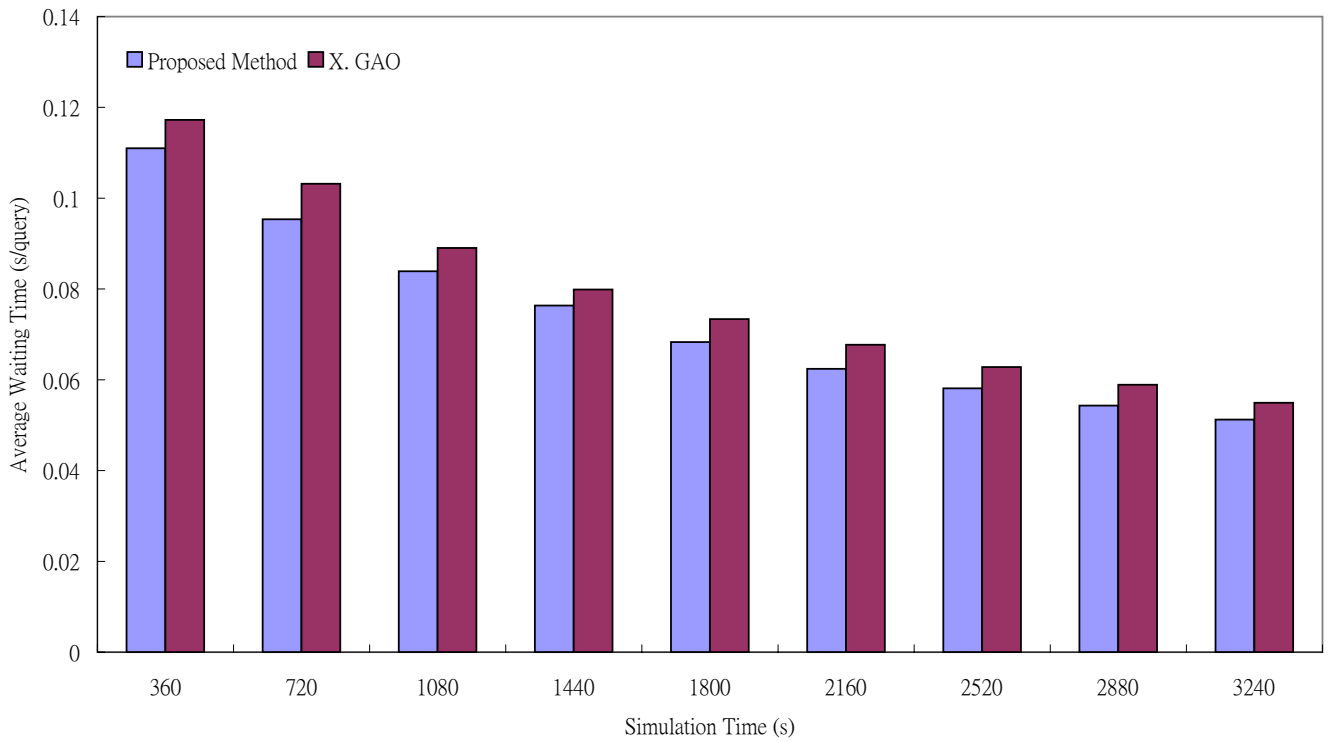


Figure 5.3: Waiting time for NN query at very start

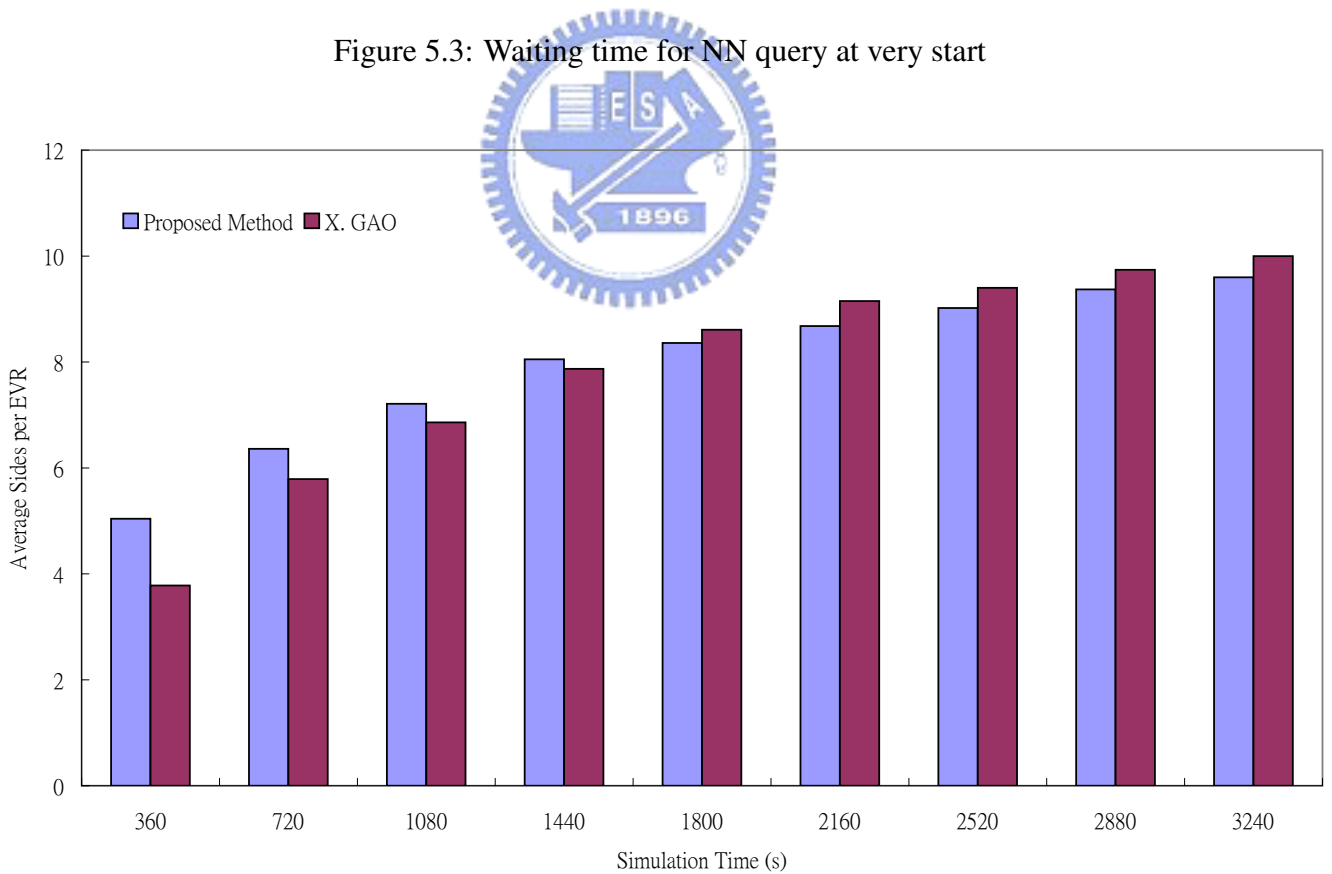


Figure 5.4: Average sides for EVR in proxy for NN query at very start

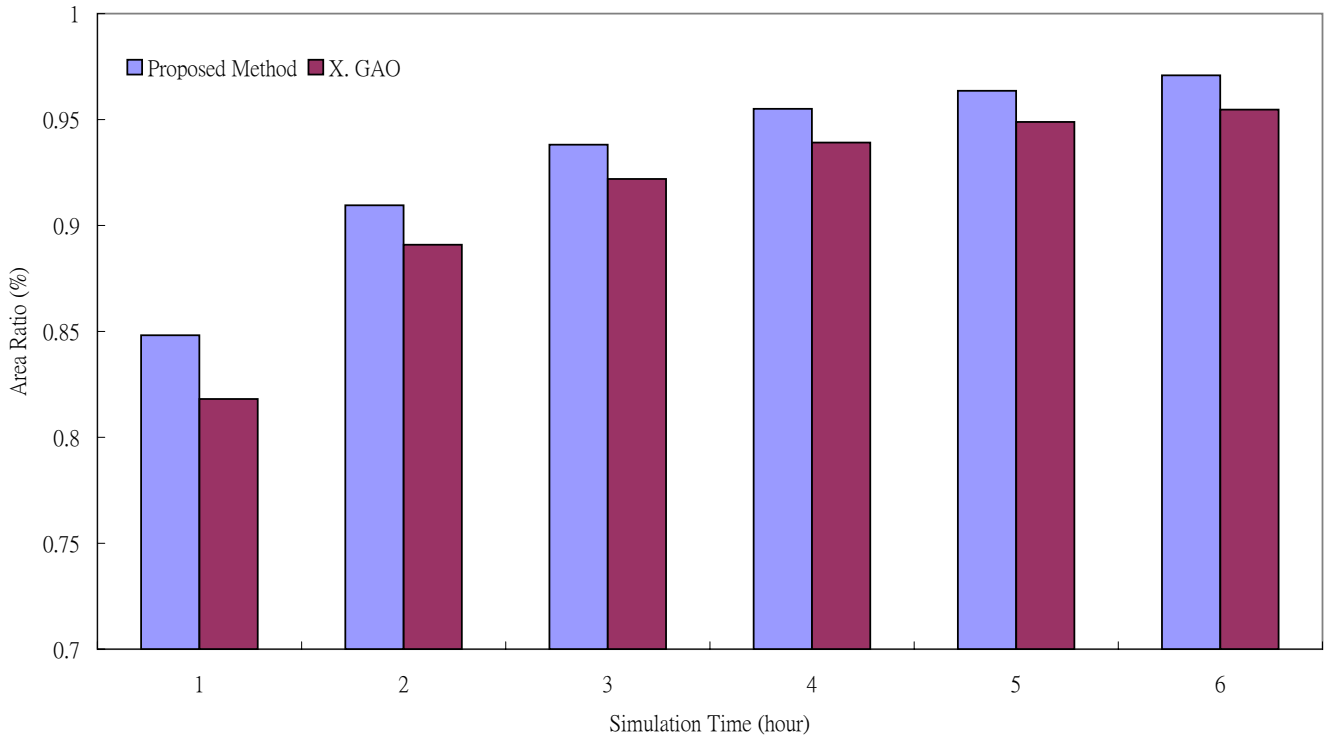


Figure 5.5: Area ratio for NN query

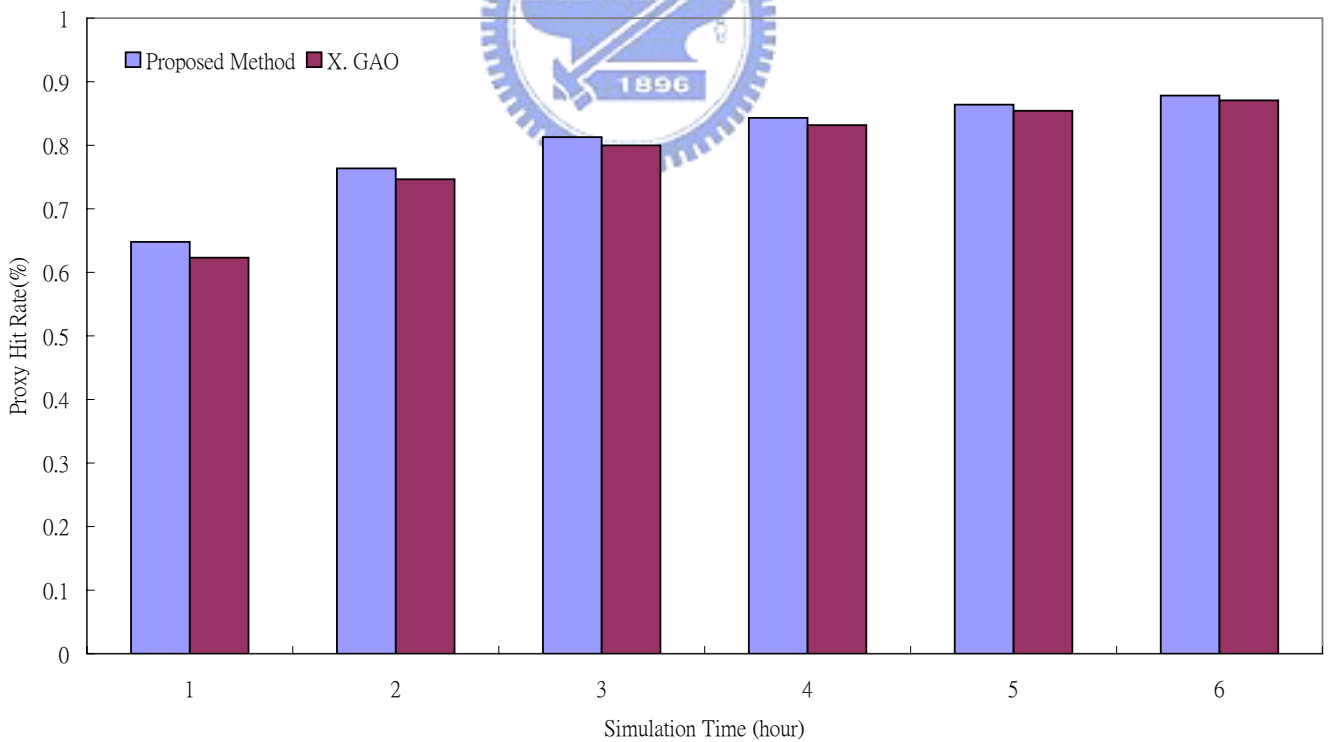


Figure 5.6: Proxy hit rate for NN query

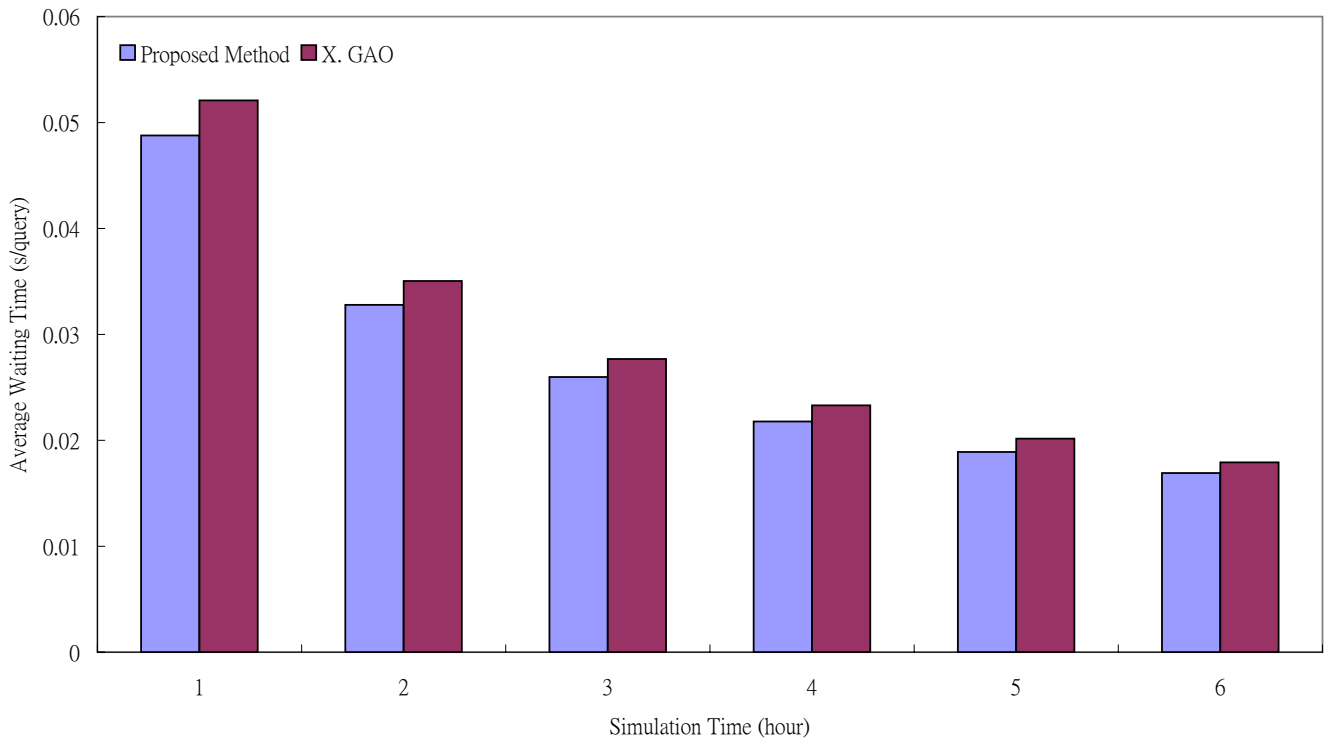


Figure 5.7: Waiting time for NN query

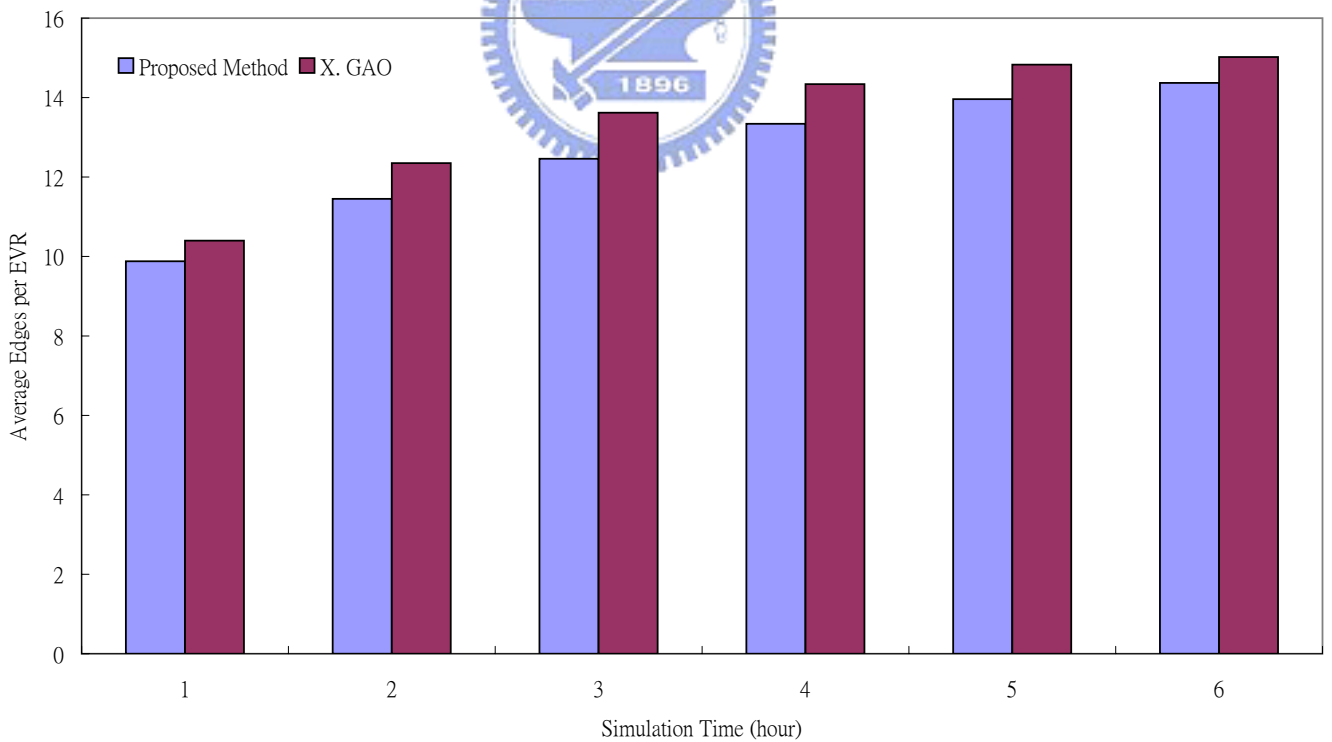


Figure 5.8: Average sides for EVR in proxy for NN query



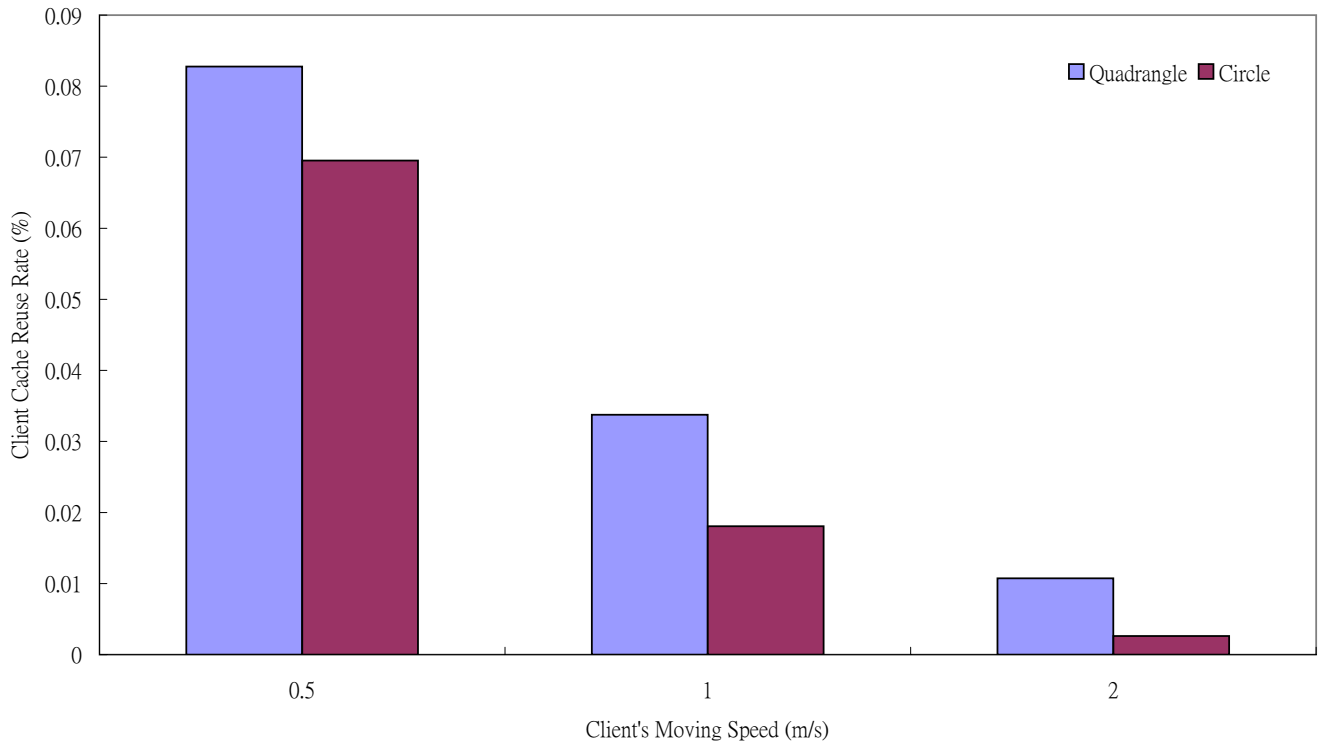


Figure 5.9: Client cache reuse

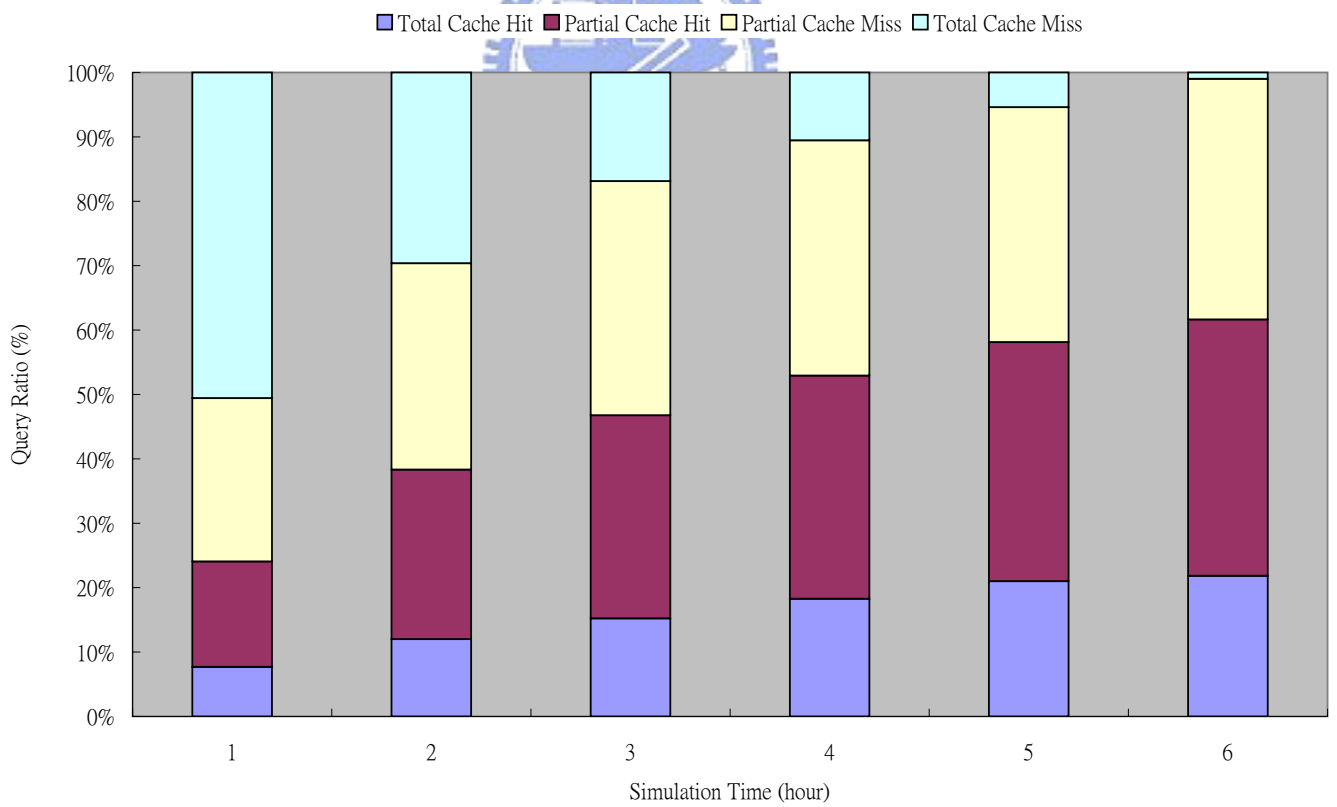


Figure 5.10: Kinds of Queries

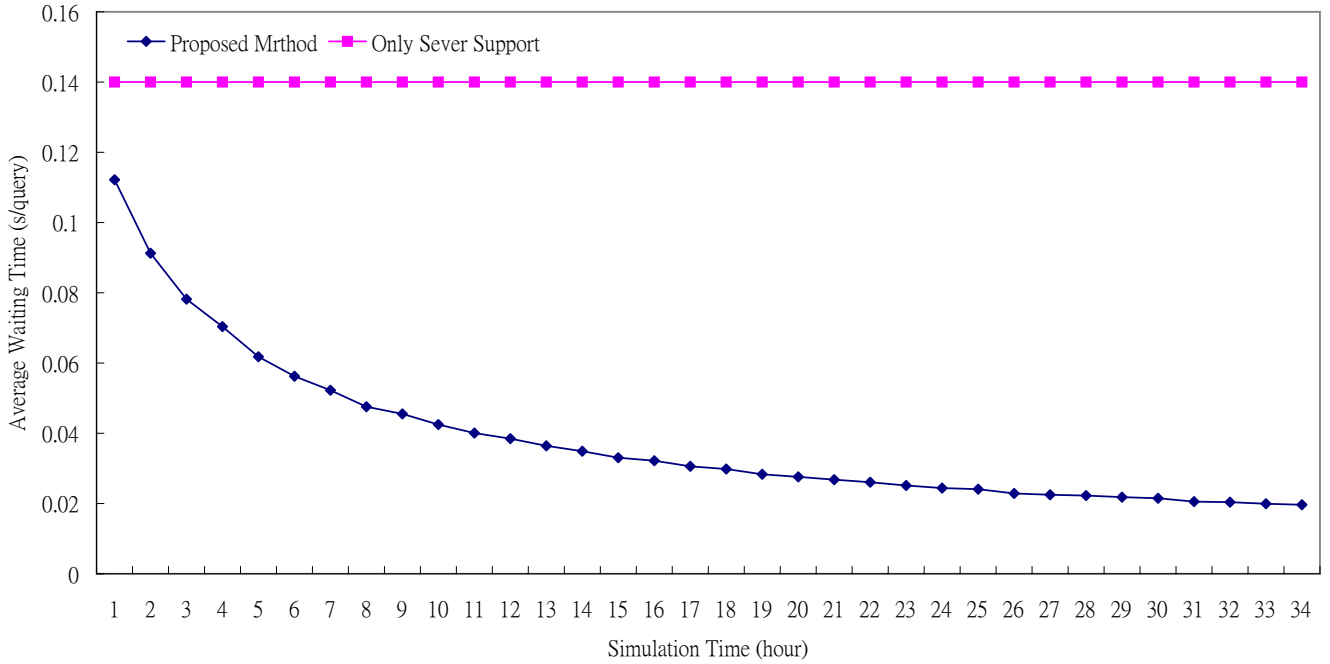


Figure 5.11: Waiting time for  $k$ NN query

each query, and average edges stored in the proxy.

We first discuss the area ratio of valid regions in proxy and AVR. Larger area ratio means much valid region area learned by proxy. Figure 5.5 shows that we can get much EVR than [5]. By this reason, Figure 5.6 shows that the proxy hit rate of our method is superior to [5]. The client in the simulation with fast-start has a lower waiting time. This is because our method organizes larger EVRs and has larger cache hit rate and fewer clients fall victim to cache misses via the proxy. Figure 5.7 shows these results. However, the cost of our method is almost the same as [5], Figure 5.8. That means we succeed in providing a suboptimal EVR combination method than [5].

### 5.2.3 Effect of Using EVR for Client

Here we will compare the relationships between the shapes of EVR for client and their reuse rates. Larger reuse rate means that client has larger chance to answer request by itself and save energy of communication with servers. Figure 5.9 shows that quadrangle will better than circle in terms of reuse rate. The reason may be that quadrangle will have larger area than circle in EVR

for client.

#### **5.2.4 Effect of kNN Situation**

Here we will discuss situations in kNN query. Figure 5.10 shows the kinds of queries. We can see that the number of total cache hit, partial cache hit and partial queries are getting more and more. Figure 5.9 shows that the average waiting time of each query of our method will get less because we can learn EVR of kNN. But [5] can do this only by server support. Thus the average waiting time of each query is constant.



# Chapter 6

## Conclusion

In this thesis we study moving NN and k-NN search problems, and propose architecture and methods for handling these problems. Despite the advantages of valid regions, in a real-world scenario, servers do not provide valid regions to clients. However, the only existing proxy method for moving NN queries is still not efficient enough in combination with EVRs at beginning of queries and partway into the process. We propose an enhanced proxy method that can provide faster response times and shorter waiting times for clients. Furthermore, the proposed method of cache replacement also handles cache space better than previous models. In addition, we extend these methods to support moving k-NN queries.

The results of the simulation show that our proposed method is superior to existing methods in response time and database workload. Our method greatly boosts system performance.

# Bibliography

- [1] C. Becker, and F. Durr. On Location Models for Ubiquitous Computing. *Personal and Ubiquitous Computing*, 2005.
- [2] T. Camp, J. Boleng, and V. Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communication and Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2002.
- [3] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004.
- [4] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 1998.
- [5] X. Gao, and A. R. Hurson. Location Dependent Query Proxy. In *Proceedings of the 2005 ACM symposium on Applied computing*, 2005.
- [6] X. Gao, J. Sustersic, and A. R. Hurson. Window Query Processing with Proxy Cache. In *Proceedings of the 7th IEEE International Conference on Mobile Data Management*, 2006.
- [7] H. Hu, J. Xu, and D. L. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *Proceedings of the ACM International Conference on Management of Data*, 2005.
- [8] H. Hu, J. Xu, W. S. Wong, B. Zheng, D. L. Lee, and W. C. Lee. Proactive Caching for Spatial Queries in Mobile Environments. In *Proceedings of the 21st IEEE International Conference on Data Engineering*, 2005.
- [9] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005.
- [10] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proceedings of 7th IEEE International Symposium*, 2001.
- [11] X. Xiong, M. F. Mokbel, and W. G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *Proceedings of the 21st IEEE International Conference of Data Engineering*, 2005.
- [12] B. Zheng and D. L. Lee. Information Dissemination via Wireless Broadcast. *Communications of the ACM*, 2005.
- [13] B. Zheng, J. Xu, and D. L. Lee. Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments. *IEEE Transactions on Computer*, 2002.
- [14] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.
- [15] JSIM: A Java-based simulation and animation environment.. <http://www.cs.uga.edu/jam/jsim/>.
- [16] Spatial Index Library. <http://research.att.com/marioh/spatialindex/>.