Supporting Advanced Interactive Search Using Inverted Index

Zheng Yuxin

(B.Eng., South China University of Technology)

Supervisor: Prof. Anthony K. H. Tung

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2016

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Zheng Yuxin

Date: _____

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor Professor Anthony K. H. Tung for his guidance and continuous support of my study. Professor Tung is a smart and creative professor, who always provides me with innovative ideas and gives insightful comments to my research. I sincerely appreciate his guidance and encouragement during my study. I will never forget the moment when our paper was rejected after a revision. He encouraged me, saying that we might just need a bit more luck. His encouragement made me move on. Without his guidance, this dissertation would not have been completed.

Besides, I would like to thank the members of my thesis committee: Professor Kian-Lee Tan and Professor Wing-Kin Sung, for their insightful comments, and suggestions to improve the quality of the thesis. I am grateful to my project supervisor Professor Beng Chin Ooi. His strictness pushes me to work hard and overcome the difficulties encountered during my study. I also appreciate the efforts from all the coauthors in the past papers, including Professor Zhifeng Bao, Mr. Qi Guo, Professor Lidan Shou, and Professor Sai Wu. I really appreciate the intense discussions to improve the quality of our papers.

I thank the members in iData Group: Ramon Bespinyowong, Bingtian Dai, Qi Guo, Wei Kang, Chen Liu, Qi Liu, Meiyu Lu, Zhan Su, Nan Wang, Xiaoli Wang, Shanshan Ying, Dongxiang Zhang, Jingbo Zhang, Zhenjie Zhang, Feng Zhao and Jingbo Zhou, for the helpful discussions and the sleepless nights before conference

deadlines. I truly feel that we are a team. Also, I thank the members in the ARShop project: Qiang Hu, Zhaoxian Li and Lifu Wu, for their hard work. In addition, I would like to express my appreciation to my lab mates and friends for all the help and support. I want to give my special thanks to my friends: Yongfeng Chen, Huihua Feng, Hao Li, Ruiming Tang, Huiyuan Wu, Qiuyong Yang and Zhijian Zhen, for being always there with me.

Last but not least, my deepest thanks goes to my parents: Xiaochuan Zheng and Ruhua Zheng, for their unconditional love and support. Their love and support give me the faith and strength to face all kinds of difficulties in my life.

Contents

Acknowledgements **v**

Summary **xi**

List of Tables **xiii**

List of Figures **xv**

1 Introduction **1**

1.1 Background and Motivation 1

1.1.1 Interactive Methods in Location-based Keyword Query 7

1.1.2 Interactive Methods in Location-based Image Query 8

1.1.3 ARShop: Interactive Augmented Reality System for Shopping 11

1.2 Gap and Purpose 12

1.3 Organization 14

2 Literature Review **17**

2.1 Query Recommendation 17

2.1.1 Quality Measurement of Query Recommendation 19

2.2 Keyword Search in Spatial Databases 21

2.2.1 Extensions of Spatial Keyword Search 22

2.3 Locality-Sensitive Hashing 24

2.3.1 Similarity Search and kNN Processing 25

2.3.2 Locality-Sensitive Hashing 26

2.3.3 Fractional Distance Metric 32

3 INSPIRE: A Framework for Incremental Spatial Prefix Query Relaxation 35

3.1 Motivation 36

3.2 The INSPIRE Framework 40

3.2.1 Preliminary 40

3.2.2 Problem Statement 42

3.2.3 Framework 45

3.3 Two-level Inverted Index 47

3.3.1 Spatial q-gram 47

3.3.2 Object-level & Node-level Inverted Index 48

3.3.3 Combining q-gram with positional q-grams 50

3.3.4 Index Implementation & Maintenance 50

3.3.5 Space Requirement 52

3.4 Query Processing 53

3.4.1 Node-level and Object-level Filters 55

3.4.2 Complexity Analysis 57

3.5 Intra-Query Optimization 58

3.5.1 Selectivity Estimation 58

3.5.2 Reuse of Query Results 61

3.6 Inter-query Optimization 63

3.6.1 Processing Appending Queries 63

3.6.2 Merging Inverted Lists for Appending Queries 65

3.7 Experiment 67

3.7.1 Experiment Setting 68

3.7.2 Comparison between Variants of `INSPIRE` 70

3.7.3 Comparison to Other Approaches . . . 74
3.7.4 Effectiveness Study . . . 78
3.8 Implementation of INSPIRE . . . 81
3.8.1 User-friendly Interface . . . 82
3.8.2 Local Query Processor . . . 82
3.8.3 System interface . . . 83
3.9 Summary . . . 85

4 LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index 87

4.1 Motivation . . . 88
4.2 LazyLSH . . . 93
4.2.1 Overview . . . 93
4.2.2 LSH in an ℓ_p Space . . . 94
4.2.3 Computing Internal Parameters . . . 98
4.3 Query Processing . . . 104
4.3.1 Processing $\mathcal{R}_p(\vec{q}, \delta, c)$ . . . 104
4.3.2 Processing $\mathcal{N}_p(\vec{q}, k, c)$ . . . 107
4.3.3 Multi-query Optimization . . . 108
4.3.4 Extending LazyLSH to Other Existing Methods . . . 109
4.4 Experiments . . . 110
4.4.1 Datasets and Queries . . . 111
4.4.2 Evaluation Metrics . . . 112
4.4.3 Study on Synthetic Datasets . . . 114
4.4.4 Study on Real Datasets . . . 116
4.5 Summary . . . 123

5 ARShop: An Interactive Shopping System using Augmented Reality 125

5.1 Motivation 126

5.2 System Architecture 130

5.2.1 Storage Layer 132

5.2.2 Index Layer 132

5.2.3 Application Layer 133

5.3 Demonstration 134

5.3.1 ARShop Website 134

5.3.2 Mobile Application 136

6 Conclusion and Future Work 139

6.1 Conclusion 139

6.2 Future Work 141

Bibliography 143

Summary

With the proliferation of geographic applications, a large amount of geographic data have been created, such as the geo-textual data and geotagged images. Meanwhile, location-based services are developed to provide people with information of their surrounding environments. Two types of location-based queries are developed for the geo-textual data and the geotagged images respectively. Recent study on spatial keyword search focused on the processing of spatial keyword queries which retrieve objects that match particular keywords within a spatial region. Besides, spatial image search is proposed to find similar images and information about the query image. However, such a huge amount of geographic data also pose a question: how can we find the exact information we want? Interactive methods have been studied widely to find the right information based on human-computer interaction. In this thesis, we focus on the location-based services and study the advanced interactive methods in these services using inverted index.

We propose a general framework to process the location-based keyword queries interactively. The proposed framework adopts a unifying strategy for processing different variants of spatial keyword queries. We adopt the autocompletion paradigm that generates the initial query as a prefix matching query. If there are few matching results, other variants of spatial keyword search are performed as a form of relaxation that reuses the processing performed in the earlier phase. The types of relaxation allowed include spatial region expansion and exact/approximate prefix/substring

matching. Moreover, since the autocompletion paradigm allows appending characters after the initial query, we look at how query processing performed for the initial query and relaxation can be reused in such instances. Compared to existing work which processes variants of spatial keyword query as new queries over different indexes, our approach offers a more compelling way to efficient and effective spatial keyword search.

For the location-based image search, it can be modeled as the nearest neighbor search in high-dimensional spaces. Due to the "curse of dimensionality" problem, it is very expensive to process the nearest neighbor (NN) query in high-dimensional spaces; and hence, approximate approaches, such as Locality-Sensitive Hashing (LSH), are widely used for their theoretical guarantees and empirical performance. Current LSH-based approaches target at the ℓ_1 and ℓ_2 spaces, while, as shown in the existing work, the fractional distance metrics (ℓ_p metrics with $0 < p < 1$) can provide more insightful results than the usual ℓ_1 and ℓ_2 metrics for data mining and multimedia applications. However, none of the existing work can support multiple fractional distance metrics using one index. We propose LazyLSH that answers approximate nearest neighbor queries for multiple ℓ_p metrics. Different from previous LSH approaches which need to build one dedicated index for every query space, LazyLSH uses a single base index to support the computations in multiple ℓ_p spaces, thereby significantly reducing the index maintenance overhead. LazyLSH can provide more accurate results for approximate kNN search under fractional distance metrics.

Finally, we design and develop an interactive augmented reality system for shopping, called ARShop, to provide the above location-based services. The main objective of our system is to enhance users' shopping experience. A user can input a shopping list and issue a location-based query using a photo of his/her surrounding environment. Based on the shopping list and the user-uploaded photo, the ARShop system can direct the user to shops that contain items in his/her shopping list.

List of Tables

3.1 Table of Notations for Chapter 3 43

3.2 Spatial Prefix Query and its Relaxation 45

3.3 Two-level Inverted Index 49

3.4 Filters used in a particular query 56

3.5 Statistics of the used data sets in Chapter 3 68

3.6 Index Construction Time and Index Size 70

3.7 Query Examples 80

4.1 Classification accuracy 89

4.2 Table of Notations for Chapter 4 92

4.3 Parameter Settings for the synthetic datasets 112

4.4 Statistics of the real datasets and index sizes 112

4.5 Index size w.r.t. different parameter settings 115

List of Figures

1.1 Interactive Search 2

1.2 Location-based Keyword Query 5

1.3 Location-based Image Query 5

1.4 Overview of ARShop 11

3.1 Spatial Prefix Query and Its Relaxation 38

3.2 Hilbert Curve at Level One 41

3.3 Hilbert Curve at Level Two 41

3.4 Hilbert-encoded Quadtree of Eleven Objects 44

3.5 The INSPIRE Framework 46

3.6 Merging Lists of Case (2) 67

3.7 Impact of the max capacity 69

3.8 Relaxation of the SP query 71

3.9 Precision and Recall 72

3.10 Impact of inter-query optimization w.r.t. appending characters 73

3.11 Impact of inter-query optimization w.r.t. query ranges 73

3.12 Comparisons of SP & SS Queries w.r.t. query ranges 74

3.13 Comparisons of SP & SS Queries w.r.t. query ranges 74

3.14 Comparisons of SAP & SAS Queries w.r.t. query ranges 76

3.15 Comparisons of SAP & SAS Queries w.r.t. edit distances 76

3.16 Efficiency 77

3.17 Effectiveness 79
3.18 System Architecture 81
3.19 Interface of INSPIRE 83
4.1 LazyLSH Overview 91
4.2 Use $B_1(\vec{q}, r)$ to approximate $B_p(\vec{q}, \delta)$ (Best viewed in color) 95
4.3 Bounds of ℓ_p distance 95
4.4 p'_1 and p'_2 for $\ell_{0.5}$ in $\mathbb{R}^{128}$, $c = 2$ 103
4.5 $(\hat{p}'_1 - \hat{p}'_2)$ w.r.t the ℓ_p spaces in $\mathbb{R}^{128}$, $c = 2$ 103
4.6 Number of functions required in $\mathbb{R}^{128}$ 103
4.7 Query-centric rehashing 106
4.8 $(\hat{p}'_1 - \hat{p}'_2)$ w.r.t d in the $\ell_{0.5}$ space 116
4.9 I/O costs on real datasets w.r.t. the ℓ_p distance 117
4.10 I/O costs on real datasets w.r.t. the number of k 118
4.11 Multiple-query optimization 119
4.12 Time with multi-query optimization 120
4.13 Average overall ratios w.r.t. c 120
4.14 Query time w.r.t. dimensionality 120
4.15 Average overall ratios on real datasets 122
4.16 Impact of query-centric rehashing 123
5.1 Overview of ARShop 130
5.2 System architecture of ARShop 131
5.3 ER diagram of ARShop 133
5.4 Overview of the Website 134
5.5 Web Interfaces 135
5.6 Mobile application for shop owners 136

5.7 Mobile application for shoppers 137

Chapter 1

Introduction

1.1 Background and Motivation

We are living in the age of big data where the amount of accessible data is almost limitless. Terabytes or even petabytes of data are being accumulated daily from a wide variety of sources such as news sites and social networks. The amount of accessible data is almost limitless. Such a huge amount of data also bring a question: how can we find the exact information we want? Interactive search methods are meant to address this problem of finding the right information based on human-computer interaction.

In search engines, the most important issue to be resolved is how to improve users' search experience. Regarding search experience, effectiveness is a critical factor in evaluating the performance of search engines. It depends on whether search engines can return the exact information that users want. In order to support effective search, interactive search is proposed to indicate users' preferences [118, 143].

It is known that information retrieval is an interactive and iterative process [115]. The idea of interactive search was first proposed in [130]. Swanson [130] pointed out that the essential role in information retrieval is a trial-and-error process, which indicates that a query needs to be formulated and refined several times to get the expected results. Later, Efthimiadis [49] identified two query stages: the initial stage

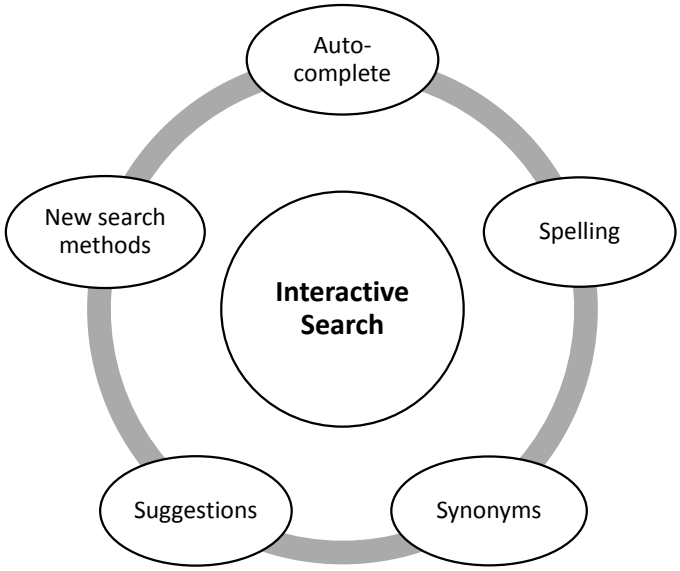


Figure 1.1: Interactive Search

and the query reformulation stage. In the initial stage, the initial query is constructed and answered. If the initial query does not return the expected search results, users can further reissue a query in the query reformulation stage. In this stage, the initial query is adjusted manually or with the assistance of the system. Such a process is referred to as the query formulation or query recommendation [7, 57].

Generally, interactive search can be divided into five categories, as presented in Figure 1.1.

1. **Autocompletion**: predicts what users might be searching for. Autocompletion is a widely used mechanism to get the desired information quickly and with as little knowledge and effort as possible [12]. It predicts what words users want to type in without typing in the words completely. A straightforward method to achieve the autocompletion function is to apply prefix search to search engines using prefix tree [44]. Subsequent methods are proposed to relax the constraint of matching the terms from the beginning, such as substring matching using suffix tree [96]. Currently autocompletion is widely used

in commercial search engines for its effectiveness.

2. **Spelling correction**: identifies and corrects spelling errors. Approximate string search is an error-tolerant design to handle the cases when the input query text or the texts in the database contain typos. It finds strings that match a query string approximately and it is usually done by returning strings with small edit distances to the query [104].

3. **Synonyms**: recognizes words with similar meanings. A synonym is a word or phrase that means exactly or nearly the same as another word or phrase in the same language. When users issue queries, answers should contain terms with similar meanings. Finding synonyms can be conducted by searching terms for determining synonyms or other replacement terms used in an information retrieval system [135]. This technique has been used by various search engines including Google [81].

4. **Query suggestions**: provides search alternatives. Query suggestions are implemented in search engines to provide better user experience to suggest relevant queries. Recently, query logs, which can be considered as a rich source of knowledge on user behaviors, have been widely used to in query suggestions. Useful information is extracted by analyzing query logs and used to suggest alternative queries [16, 75, 76].

5. **New search methods**: creates new ways to search, e.g. "search by image" and "voice search". Conventional keyword search might not be used easily to describe complicated objects such images or audio. In order to facilitate searching such complicated objects, new search methods are proposed, such as search by image [45, 95, 51, 9, 107] and search a song by only a part of it [101, 78].

With the proliferation of geographic applications, a large amount of geographic data have been created, and the geographically encoded data are ubiquitous these days. For example, business owners can easily create and register their businesses online using Google Places for Business[1]. Rich information about a company, such as the company name, address, descriptions and contact numbers, can be included. Besides the abundant geo-textual data, a large number of photos and videos are created together with geographical locations using online map services such as Flickr and Panoramiom. Such a large amount of geographic data pose a challenge of storing and retrieving the geographic data efficiently and effectively.

In the meanwhile, there are plenty of location-based services developed because GPS-embedded smart phones are widely used nowadays. Location-based services are defined as services that are enhanced by the location information [46]. Commercial location-based services include Google Maps[2], Foursquare[3] and Facebook Places[4]. They use the location functionalities of mobile phones, such as the GPS and WiFi positioning, to provide people with information of their surrounding environments. Questions about the surrounding environments can be answered using location-based services. We proceed to illustrate two types of location-based questions with examples.

- **Location-based Keyword Query:** An example of the location-based keyword query is "Where is the nearest restaurant or ATM machine?" Users can issue a location-based keyword query by specifying a query location and a set of keywords such as "restaurant" and "ATM machine".

- **Location-based Image Query:** An example of the location-based image

[1] http://www.google.com/business/placesforbusiness/

[2] https://maps.google.com/

[3] https://foursquare.com/

[4] https://www.facebook.com/about/location/

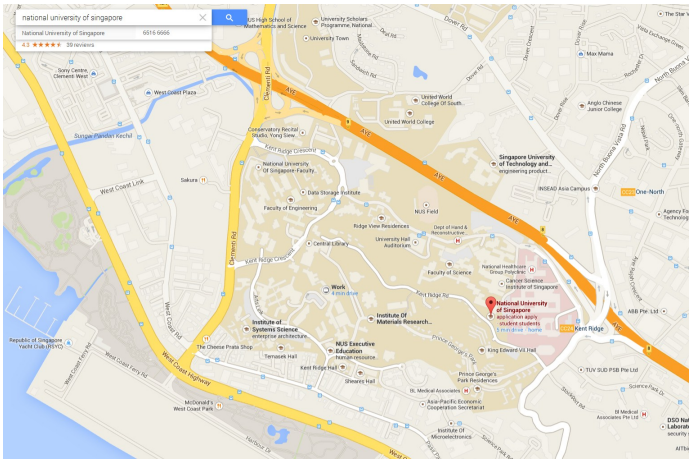


Figure 1.2: Location-based Keyword Query

query is "What is the tourist attraction in front of me?" Users can issue a location-based image query by specifying a query location and an image of the surrounding environment.

Figure 1.3: Location-based Image Query

One type of location-based services is the location-based keyword search, as shown in Figure 1.2. Users type in a set of keywords and specify a spatial area to issue a query. It is a special case of keyword search, in which the spatial information is also considered. Keyword search is a practical information retrieval mechanism to retrieve data on the World Wide Web effectively. It is commonly used because of its

user-friendly interface. In addition, it provides the functionality that users do not need to know either a query language or the underlying structure of the data. It is widely used in commercial search engines on top of documents to find the query terms. Typically, a keyword is associated with a document when the keyword is contained in the document. When users type in a set of keywords, search engines usually return the documents that are associated with the query keywords. In this thesis, we combine autocompletion and spelling correction in the context of location-based keyword search as a type of interactive search.

Another type of location-based services is the location-based image query, as shown in Figure 1.3. Users submit images of the surrounding environments as queries, which can be considered as image search. Image search is introduced to find related images from the Web [52, 86], and it can be regarded as the nearest neighbor search in high-dimensional space as images are usually represented as high-dimensional vectors [139, 80]. It has been integrated into commercial search engines, such as Google Image search[5], to improve the effectiveness of the search. We then study the location-based image search as a new search method in interactive search.

Besides effectiveness, efficiency is another important issue for search engines. In order to support fast query processing, many index structures have been proposed over the past decades. In particular, the inverted index structure is widely used in many applications, such as keyword search [154], graph search [34] and image search [32], for its simplicity and efficiency. A typical inverted index structure is represented as a collection of lists. Each list is a key-value pair, in which the key is a term and the value is a list of object identifiers containing the term. When a query comes, we first extract the terms that the query has. Then we can easily retrieve the identifiers of the objects containing those terms using an inverted index.

In this thesis, we focus on the advanced interactive techniques utilized in the

[5] http://www.google.com/imghp/

context of location-based services. In particular, based on the two types of location-based queries, advanced interactive methods towards effective search are presented. In order to achieve real-time human-computer interaction, we propose new index structures, which are based on inverted indexes, to support the advanced interactive search.

1.1.1 Interactive Methods in Location-based Keyword Query

Keyword search is receiving increasing interest in the context of spatial databases recently, namely spatial keyword query. Typically, given a spatial region and a set of keywords, spatial keyword search returns points of interest (POIs) locating in the spatial region and containing all the keywords [65]. To achieve this goal, a range of techniques have been proposed to efficiently process spatial keyword queries [33].

Spatial keyword search is generally used in map services to return POIs. It has received much attention in the research community recently [33]. In order to ensure effective retrieval, interactive methods were proposed to spatial keyword search, including autocompletion [13, 74], and the allowance of errors in keyword matching [140, 4].

On the one hand, autocompletion [12, 126] is a feature that predicts what users want to type in without typing in the words completely. It is effective and widely used in commercial search engines and integrated into spatial databases recently. In [13], the spatial prefix query is proposed to implement autocompletion in spatial keyword search. A POI is returned if it is located in the query area and the query text is a prefix of its textual content.

On the other hand, the allowance of errors in keyword matching is the technique of finding strings that match a query string approximately [99]. It is an error-tolerant design to handle the cases when the input query text contains typos. While such extensions [13, 74, 140, 4] are useful in their own right, they adopt different

indexes and query processing methods, making it difficult to incorporate all of these extensions in a single system.

Having observed that different types of relaxation have different levels of complexity and have dependencies among themselves, we propose a general framework, called INSPIRE, for processing the spatial prefix query and its relaxation. We adopt a unifying strategy for processing different variants of spatial keyword query. We adopt the autocompletion paradigm that generates the initial query as a prefix matching query. If there are few matching results, other variants of the spatial keyword search are performed as a form of relaxation that reuses the processing done in the earlier phase. The types of relaxation allowed include spatial region expansion and exact/approximate prefix/substring matching.

If no or few results are returned, we incrementally process different types of relaxation until sufficient results are returned. In particular, the spatial region of the prefix query is first expanded, followed by relaxation of different degrees on string dimension: the substring query, the approximate prefix query and the approximate substring query. As a result, our approach can support fuzzy type-ahead search in the context of spatial keyword query, which provides real-time human-computer-interaction for users. Our approach can be viewed as a combination of autocompletion and spelling correction in interactive search.

1.1.2 Interactive Methods in Location-based Image Query

In addition to text data, geographically encoded images are becoming more and more common. Users can easily create, tag and share customized photos with the help of image sharing services, such as Flickr and Panoramiom. In location-based image queries, one typical application is landmark recognition. Landmark recognition has become an active research topic in the last decade. It determines where a photo is taken based on the image database collected from the Web. It is useful in many

applications such as location recognition [69] and photo tourism [128, 150]. It can be viewed as a type of new search method in interactive search.

In landmark recognition, the nearest neighbor (NN) search is used to retrieve similar images of the query image. The input of the nearest neighbor search is a geo-encoded image, which can be considered as a high-dimensional vector. Compared to textual data, a high-dimensional vector representing an image can be viewed as a "keyword". Therefore, the nearest neighbor search on high-dimensional data used in landmark recognition can be considered as a special case of spatial keyword search.

State-of-the-art kNN processing techniques have been proposed for low-dimensional cases. However, due to the "curse of dimensionality", the same techniques cannot be directly applied to high-dimensional spaces. It was shown that conventional kNN processing approaches become even slower than the naive linear-scan approach [42]. One compromise solution is to adopt the approximate kNN technique which returns k points within distance cR from a query point, where c is the approximation ratio and R is the distance between the query point and its true (k)th nearest neighbor. The intuition is that in high-dimensional spaces, approximate results with error bounds are good enough for most applications.

To process approximate kNN queries, several methods have been proposed [8, 68, 3, 98], among which, locality-sensitive hashing (LSH) [68] is widely used for its theoretical guarantees and empirical performance. In essence, the LSH scheme is based on a set of hash functions from the *locality-sensitive hash family* which guarantees that similar points are hashed into the same buckets with higher probabilities than dissimilar points. The LSH scheme was first proposed by Indyk and Motwani et al. [68] for the use in the binary Hamming space, and later was extended for the use in the Euclidean space by Datar et al. [42] based on the p-stable distribution (e.g., the *Cauchy distribution* is 1-stable and the *Gaussian distribution* is 2-stable).

The effectiveness of high-dimensional applications was shown to be sensitive to

the choice of distance functions [2]. Although the Manhattan (ℓ_1) and Euclidean (ℓ_2) metrics can reveal some important properties of the data, ℓ_p metrics with $0 < p < 1$, i.e., *fractional distance metrics*, can provide more insightful results from both the theoretical and empirical perspectives for content-based image retrievals [67] in high-dimensional spaces.

This brings us a challenging problem: how to answer approximate kNN queries under multiple fractional distance metrics. A naive method is to build an LSH index for every possible p value, which is obviously too expensive. Moreover, due to the lack of closed form density and distribution functions for p-stable distributions when $p \neq 1$ or 2, it is not trivial to generate p-stable random variables and build an optimal index structure for the fractional distance metrics. To address the problem, we propose LazyLSH as an efficient mechanism to process approximate kNN queries in different ℓ_p spaces.

LazyLSH builds an LSH index in a predefined ℓ_p space, which is referred to as the base space. Using this materialized index, LazyLSH can answer approximate kNN queries in a user-specific query space. The word "Lazy" is borrowed from the lazy learning algorithms [145] in which generalization beyond the training data is delayed until a query is issued. LazyLSH means that we do not build an index for every query space. Instead, we reuse the index constructed in the base space to answer queries in the user-specific query space. Our analysis shows that if two points are close in an ℓ_{p1} space, then they are likely to be close in another ℓ_{p2} space. We also find that a locality-sensitive hash function built in the base space is still locality-sensitive in the query space when certain conditions hold. With this observation, LazyLSH can answer approximate kNN queries in different ℓ_p spaces to support the processing of location-based image queries under different distance functions.

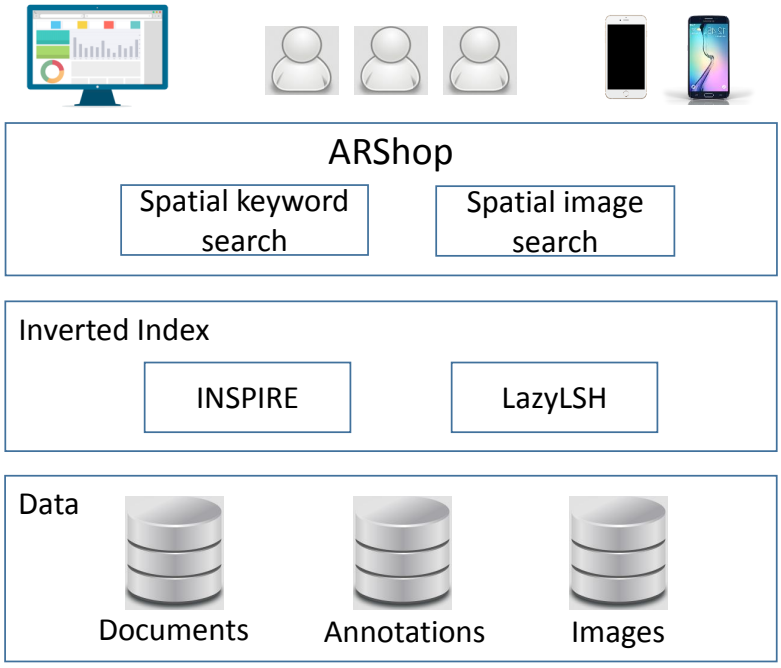


Figure 1.4: Overview of ARShop

1.1.3 ARShop: Interactive Augmented Reality System for Shopping

We develop a system, called ARShop, to utilize the above interactive methods in location-based services. The main objective of ARShop is to enhance the shopping experience for users. This system can also be used for tourist navigation in amusement parks such as Disneyland and Universal Studios.

First, ARShop provides shop owners with cloud-based tools to digitize their shops. A shop owner can create a digital shop and upload images of the shop. The shop owner can further add annotations on their uploaded photos to indicate the shop's name and available products in the shop.

For shoppers, ARShop provides them with a tool to input their shopping lists. A user can submit a shopping list or a photo of his/her surrounding environment to issue a query. Then our system can direct the user to find the items in the shopping list. The system supports two primary functions, which is shown in Figure 1.4:

- Locating shops that contain items in a user's shopping list. This function is done by performing spatial keyword queries. The user's shopping list is taken as queries and used to match with the annotations from the shop owners. As a result, we can find the shops that contain items in a user's shopping list.

- Locating the user's position. This function is done by image matching as GPS may not be accurate when used indoor. The photo of his/her surrounding environment is taken as a query and used to compare with the nearby images. Therefore, the ARShop system can infer the current location of the user, and direct the user to the shops that contain items in the user's shopping list.

1.2 Gap and Purpose

After reviewing the existing work on interactive methods in the context of location-based services, we summarize some research gaps for the current study as follows.

1. While the extensions of spatial keyword search are useful in their own right, they adopt different indexes and query processing approaches, making it difficult to incorporate all of these extensions in a single system. To address this problem, we aim to develop a general framework for integrating these extensions of spatial keyword search to provide real-time fuzzy type-ahead search on geo-textual data.

2. The approximate nearest neighbor search used in spatial image search has been well studied for the Euclidean distance in high dimensional space. However, none of the existing work can support multiple fractional distance metrics using one index. To address this problem, we aim to develop a method to answer approximate nearest neighbor queries under multiple fractional distance metrics using one index.

The aim of this study is to develop interactive search techniques to facilitate searches in location-based services. The specific objectives of this research are to:

1. Develop a general framework for the interactive location-based keyword search, which allows the processing of the extended spatial keyword queries. Under this framework, we order and perform the query relaxation incrementally, starting from the least expensive to the most. This is done by ensuring that the relaxation performed at later stages can reuse the results from the earlier query processing and relaxation as much as possible.

2. Develop an index structure that builds an LSH index in a predefined ℓ_p space. Based on the materialized index, approximate kNN queries under multiple fractional distance metrics can be answered using this single materialized LSH index. This technique can be used to support image search, which can be viewed as a new type of search method in interactive search.

3. Build an interactive augmented reality system for shopping. This system can support the spatial keyword search and spatial image search. By integrating these two location-based services, this system can infer the current location of a user and direct the user to shops that contain items in the user's shopping list.

The theoretical frameworks and devised algorithms of this study may have significant impacts on two aspects:

1. Different types of relaxation are processed under a unified framework. In this case, common index structures can be used and results obtained at earlier steps can be reused at later steps. The incremental solution may be significant because it can be extended to other applications.

2. Approximate kNN queries in different ℓ_p spaces can be answered using a single materialized LSH index. In addition, our method can also process multiple queries in different ℓ_p spaces simultaneously. Therefore, such a method can help us explore high-dimensional data and may provide insights into various applications such as classifications and clusterings.

Our work on the interactive spatial keyword search was published in [148]. Its corresponding demo was published in [147]. The work on the approximate nearest neighbor search to support spatial image queries will be published in [149]. Besides, the ARShop System can be found in http://shopbyar.com/ARShop/.

1.3 Organization

The rest of this thesis is organized as follows:

- Chapter 2 reviews the related work. The surveyed topics include interactive methods used in text searches and augmented reality techniques.

- Chapter 3 presents our work on the interactive spatial keyword search. A framework, named INSPIRE, is proposed to combine autocompletion with fuzzy search to answer location-based keyword queries to improve user experiences. A system which performs query relaxation on spatial prefix query is demonstrated.

- Chapter 4 presents a foundation to support spatial image search. A method, called LazyLSH, is presented to answer approximate nearest neighbor queries for multiple ℓ_p metrics using one inverted index.

- Chapter 5 presents our ARShop system that uses the augmented reality technology to enhance the users' shopping experience. By integrating spatial key-

word search and spatial image search, ARShop can direct users to the shops that contain items in a user's shopping list.

- In Chapter 6, we conclude this thesis and discuss possible future extensions of the current work.

Chapter 2

Literature Review

In this chapter, a literature review over advanced interactive search is conducted. We mainly focus on the topics of spatial keyword search and image search, which are typical applications of interactive search. First, existing work of query recommendation to facilitate interactive keyword search is surveyed. Then, we narrow the scope of interactive keyword search in the context of spatial databases in which the existing work of spatial keyword search and its extensions is reviewed. Finally, we review existing work of locality-sensitive hashing to perform approximate neighbor queries, which is used in spatial image search.

2.1 Query Recommendation

Web search queries issued by users are often short, typically with only one or two keywords each [127]. To assist users in formulating queries, modern search engines are equipped with machine intelligence to provide interactive search such as query recommendation [25, 63, 97, 94, 89]. In this section, the relative work of query recommendation is reviewed. First, query recommendation methods are highlighted. Then the query measurement of recommended queries is discussed.

In order to improve the quality and effectiveness of keyword search, query recommendation has been extensively studied in the past decades. Query recommendation

is implemented in search engines to provide better user experience to recommend relevant queries. Recently, query logs have been widely used to in query recommendation. Query logs are considered as a rich source of knowledge on user behaviors. The main challenge in analyzing query logs lies in extracting useful relations from the raw lists of user actions. Many different methods have been proposed to extract useful relations in query logs. Among those methods, term co-occurrence [16, 76], query clickthrough [75, 41, 111, 97], and query chains [110, 61, 62] are the most used types of information in query logs when recommending a query.

Term co-occurrence [16, 76] is a technique that measure the similarity between terms. Terms with a high number of co-occurrence are relevant to each other. By capturing the similarity between query terms, the terms that are relevant to the original query can be recommended to users.

Most of the proposals use query logs that contain query click information [75, 41, 111, 97]. By representing the relationship between queries and URLs into a bipartite click graph, many researchers have investigated techniques, such as random walks and hitting time of queries, for learning the underlying query-document relevance.

When queries are considered independently, log files only provide implicit feedback on a few results at the top of the result set for each query because users very rarely look further down the list. Different from the click-though methods, query chain based methods take the advantage of user intelligence in reformulating queries [110, 61, 62]. Query chain based methods improve from the click-though methods by useing a sequence of relative queries instead of one query only. Another benefit of using query chains is that relevant judgments can also be deduced on the many more documents seen during the entire search session.

2.1.1 Quality Measurement of Query Recommendation

In this subsection, the quality measurement of the query recommendation is presented. The quality of the recommended terms is a major concern for a query recommendation system, which reflects whether the recommended terms can actually deduce the search intent of the user. Regarding the search quality, a good query recommendation system should have the following properties [89].

1. **Relevancy:** Recommended queries should be semantically relevant to the original query.

2. **Diversity:** The recommendation should cover search intents of different interpretations of the original query.

3. **Ranking:** Higher relevant queries should be ranked ahead of less relevant ones in the recommendation list.

Based on these properties, some representative solutions are highlighted.

2.1.1.1 Query Relevancy

Semantic relevance among information resources can play an effective role in information retrieval, and there are several different approaches to measure semantic similarities. In the context of query recommendation, there are a number of studies based on query logs [75, 41, 111, 97]. Many of these studies derive click-through bipartite graph from query logs to measure query relevancy. Based on the assumption that the URLs a user clicked reflect his search intent, queries that share more common clicked URLs are more similar.

Another direction to retrieve query similarity is mining query concepts [114, 21, 35, 53, 102]. The search terms, whose concept is similar to the concept of the original query, are recommended to the user. The concepts can be extracted explicitly or

implicitly. In [114, 21, 35, 102], concepts are derived explicitly by building ontology-based taxonomy. On the other hand, in [53], concepts are extracted implicitly from the query relation graphs, which is from query relations mined using association rules. Queries that frequently co-occur in the same sessions are considered highly associated, and so they are connected in the graph. Subsets of queries that are strongly connected in the graph are taken as concepts.

Although query relevancy ensures that recommended items are similar to the original query, it may produce many redundant items. To address this problem, result diversification is introduced.

2.1.1.2 Result Diversification

Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommender systems and Web search. The basic idea of result diversification is that search results should be not only relevant to the user query, but also relevant and different from each other.

Many different approaches have been proposed in the related literature for the diversification problem. Recently, Drosou et al. [48] surveyed the methods in result diversification and summarized three types of approaches to result diversification. They are (1) content-based approach, i.e. results that are dissimilar to each other, (2) novelty-based approach, i.e. results that contain new information when compared to previously seen ones, and (3) coverage-based approach, i.e. results that belong to different categories. The authors conducted the first all-around evaluation of the existing work on result diversification. Various definitions of the result diversification problem are surveyed.

Although result diversification ensures that recommended items are diversified, the recommended items are equally weighted. In this case, popular items do not have high weights. To address this problem, result ranking is introduced to ensures that

higher relevant queries are ranked ahead of less relevant ones in the recommendation list.

2.1.1.3 Result Ranking

Ranking is an important component in search engines to show the most relevant results in front of the recommendation list [151, 29]. Recently, diversity is used along with some other ranking criterion, most commonly that of relevance to the users query. In this case, diversity and relevance are both considered in the result ranking. Therefore, the advantages of diversity and relevance can be both reserved in the result ranking.

To the best of our knowledge, the first work in which the two measures were combined is [27], in which marginal relevance, i.e. a linear combination of relevance and diversity, is proposed as a criterion for ranking results retrieved by IR systems. After that, Gollapudi et al. [58] considered the combination of relevance and diversity and presented eight intuitive axioms that diversification systems should satisfy. However, it is shown that not all of them can be satisfied simultaneously. The combination of these two criteria has also been studied in [144] as an optimization problem.

2.2 Keyword Search in Spatial Databases

In the previous section, we survey the query recommendation as an interactive method used in keyword search. In this section, we narrow the scope of interactive keyword search in the context of spatial database. Existing work on spatial keyword search and its various extensions is highlighted.

Keyword search in spatial databases has been receiving significant attention recently [152, 65, 43, 38, 116, 141, 142, 26, 33]. It is considered as a combination of spatial query and keyword search. It contains both spatial and textual constraints. In order to process the spatial keyword search efficiently, various hybrid index struc-

tures have been proposed to support the spatial keyword search.

Recently, a survey paper on spatial keyword search was published. Chen et al. [33] surveyed the spatial keyword search and summarized three types of general spatial keyword queries. They are (1) boolean kNN query, (2) top-k kNN query, and (3) boolean range query. These three types of spatial keyword queries can be illustrated with examples.

Example 2.1 (boolean kNN query)**.** *Retrieve the k objects nearest to the user's location such that each objects text description contains the keywords "coffee", "western", and "restaurant".*

Example 2.2 (top-k kNN query)**.** *Retrieve the k objects with the highest ranking scores, measured as a combination of their distances to the query location and the relevance of their text description to the query keywords "coffee", "western", and "restaurant".*

Example 2.3 (boolean range query)**.** *Retrieve all objects whose text descriptions contain the keywords "coffee", "western", and "restaurant" and whose locations are within 2 km of the query location.*

The authors conducted the first all-around evaluation of the existing work on spatial keyword search. Detailed experimental evaluation on different parameter settings was performed for the three types of common spatial keyword queries. By surveying and subjecting existing spatial textual indexing techniques, the paper offered structure that may help the area of spatial keyword search progress more effectively.

2.2.1 Extensions of Spatial Keyword Search

In the subsection above, the existing work of traditional spatial keyword search is discussed. Next the techniques to facilitate spatial keyword search are reviewed. In

order to ensure effective and user-friendly retrieval, various extensions were done to spatial keyword search including autocompletion [13, 74] and the allowance of errors in keyword matching [140, 4].

On the one hand, autocompletion is a widely used mechanism to get the desired information quickly and with as little knowledge and effort as possible [12]. It predicts what words users want to type in without typing in the words completely. It is effective and is widely used in commercial search engines and integrated into spatial databases recently. A straightforward method to achieve the autocompletion function is to apply prefix search to search engines. Prefix search is proposed and integrated to spatial databases, in order to support the autocompletion function. Roy et al. [13] extended the prefix search over spatial databases. The textual attribute is modeled as a full string. A spatial object is returned if it is located in the query region, and the query string is a prefix of the object's textual content. Ji et al. [74] conducted the location-based instant search. Unlike [13], they modeled the textual content as a set of keywords. If the textual content of an object matched all the input keywords and contains a term where the last query keyword is its prefix, it is returned to the user.

On the other hand, approximate string search is a technique for finding strings that match a query string approximately. Approximate string search is an error-tolerant design to handle the cases when the input query text contains typos. It has been studied recently in the context of spatial databases. Yao et al. [140] studied the approximate string search in spatial databases. The authors used the similarity between the min-wise signature of a node and the query string to prune branches in the tree. However, the method may produce false negatives because of the probabilistic nature of the min-wise signature. Alsubaiee et al. [4] further investigated this problem to ensure that all the results are returned. However, these methods are proposed at the word level, which require users to type at least one full

word in a query. As a result, the approximate query cannot be supported in the location-based instant search.

Besides the standalone autocompletion and approximate string search, Chaudhuri et al.[31] further argued that search engines should support autocompletion and error tolerance at the same time. The authors proposed efficient solutions to extend error-tolerant functions over prefix queries in document searches. To the best of our knowledge, supporting error-tolerant autocompletion has not yet been studied in spatial databases. This remains a gap for our study to investigate efficiently supporting error-tolerant autocompletion in spatial keyword search. Besides, the above-mentioned extensions [13, 74, 140, 4] are useful in their own right. However, they adopted different indexes and query processing approaches, making it difficult to incorporate all of these extensions in a single system. In the next chapter, a system, called INSPIRE, is presented to support these extensions of spatial keyword search.

2.3 Locality-Sensitive Hashing

In the previous sections, we survey the iterative search methods in the context of geo-textual databases. However, there are cases that textual information is not applicable such as the iterative search in augmented reality applications, in which queries are images instead of keywords. Krevelen and Poelman [134] recently surveyed the state of technologies, applications and limitations related to augmented reality. It offers a comprehensive overview of the augmented reality and provides a suitable starting point to this field.

To facilitate finding similar images in an image database, images are represented as high-dimensional visual features such as color histograms, Gist features [47] and bag-of-word features [73] using SIFT [91], SURF [14] descriptions. In augmented

reality applications, one essential operation is to find the similar features of the query feature. This operation is basically the nearest neighbor query in high-dimensional spaces. In this section, we review techniques of the nearest neighbor search in high-dimensional spaces.

2.3.1 Similarity Search and kNN Processing

In this thesis, we mainly focus on the ℓ_p distance and we begin with the definition of the ℓ_p distance.

Definition 2.1 (ℓ_p distance)**.** *The distance between any two d-dimensional points $\vec{o}$ and $\vec{q}$ in the ℓ_p space, denoted as $\ell_p(\vec{o}, \vec{q})$, is computed as:*

$$\ell_p(\vec{o}, \vec{q}) = \sqrt[p]{\sum_{i=1}^{d} |o_i - q_i|^p} \tag{2.1}$$

In particular, $\ell_p(\vec{o}, \vec{q})$ returns the Manhattan distance and Euclidean distance when p equals to 1 and 2 respectively. If $0 < p < 1$, $\ell_p(\vec{o}, \vec{q})$ is called the *fractional distance metric* [2]. In similarity search, if $\ell_p(\vec{o}, \vec{q}) \leqslant r$, we say the point $\vec{o}$ is within the *ball* of radius r centered at the point $\vec{q}$, denoted as $B_p(\vec{q}, r)$.

Definition 2.2 (Ball $B_p(\vec{q}, r)$)**.** *Given a point $\vec{q} \in \mathbb{R}^d$, and a radius r, the ball of radius r centered at point $\vec{q}$ in ℓ_p space is defined as $B_p(\vec{q}, r) = \{\vec{v} \in \mathbb{R}^d | \ell_p(\vec{v}, \vec{q}) \leqslant r\}$.*

The similarity search problem is of fundamental importance to a variety of applications such as classification [40], clustering [112], semi-supervised learning [153], collaborative filtering [123] and near-duplicate detection [18]. Given a query object q_o represented as a high-dimensional vector, a typical similarity search retrieves the k-nearest neighbors (kNNs) of q_o using a specific distance function.

kNN search has been well studied in low-dimensional spaces [117, 121]. However, retrieving the exact results becomes a nontrivial problem when the dimensionality

increases due to the "curse of dimensionality". It has been shown that the average query time with an optimal indexing scheme is superpolynomial with the dimensionality [108]. When the dimensionality is sufficiently large, conventional kNN search approach becomes even slower than the linear scan approach [42]. To address the problem, approximate nearest neighbor search is introduced as an alternative solution, which trades off the accuracy to speed up the search process. Formally, the approximate nearest neighbor query is defined as follows:

Definition 2.3 ($\mathcal{N}_p(\vec{q}, k, c)$ problem)**.** *Given a dataset $\mathcal{D}$, a point $\vec{q} \in \mathbb{R}^d$, a cardinality k, an approximate ratio c, and an ℓ_p space, the c-approximate k nearest neighbors search returns a set of k points $\mathcal{N}_p(\vec{q}, k, c) = \{\vec{o_1}, \ldots, \vec{o_k}\}$, where points are sorted in ascending order of their distances to $\vec{q}$ in the ℓ_p space, and $\vec{o_i}$ is a c-approximation of the real i^{th} nearest neighbor. Let $\vec{o_1^*}, \ldots, \vec{o_k^*}$ be the real kNNs in ascending order of their distances to $\vec{q}$. Then $\ell_p(\vec{o_i}, \vec{q}) \leqslant (c \times \ell_p(\vec{o_i^*}, \vec{q}))$ holds for all $i \in [1, k]$.*

The $\mathcal{N}_p(\vec{q}, k, c)$ problem can be answered by issuing a set of approximate range queries with increasing radii. The approximate range query is defined as:

Definition 2.4 ($\mathcal{R}_p(\vec{q},\ r,\ c)$ problem)**.** *Given a dataset $\mathcal{D}$, a query point $\vec{q} \in \mathbb{R}^d$, $\mathcal{R}_p(\vec{q},\ r,\ c)$ returns a point $\vec{o'} \in \mathcal{D}$, where $\vec{o'} \in B_p(\vec{q}, cr)$, if there exists a point $\vec{o} \in B_p(\vec{q}, r)$.*

2.3.2 Locality-Sensitive Hashing

One typical solution is to employ the Locality-Sensitive Hashing (LSH) because of its precise theoretical guarantees and empirical performance. LSH is first introduced by Indyk and Motwani [68]. It tries to map close points to the same hash bucket. Let $\mathcal{H}$ be a family of functions mapping $\mathbb{R}^d$ to some universe U. For any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$, consider a process in which we choose a function h from $\mathcal{H}$ at random, and

analyze the probability of $h(\vec{o}) = h(\vec{q})$. The family $\mathcal{H}$ is called locality-sensitive if it satisfies the following conditions.

Definition 2.5 (Locality-sensitive hashing)**.** *Let $d(\cdot,\cdot)$ be a distance function of a metric space. A family $\mathcal{H}$ is called (r, cr, p_1, p_2)-sensitive if for any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$, satisfying*

(1) if $d(\vec{o}, \vec{q}) \leqslant r$, then $Pr_{\mathcal{H}}[h(\vec{o}) = h(\vec{q})] \geqslant p_1$,

(2) if $d(\vec{o}, \vec{q}) > cr$, then $Pr_{\mathcal{H}}[h(\vec{o}) = h(\vec{q})] < p_2$,

(3) $c > 1$, and

(4) $p_1 > p_2$.

Various LSH families have been discovered for different distance metrics [6], such as the Hamming distance between binary vectors [68], the Jaccard distance between sets [22, 20], the arccos distance measuring the angles between vectors [28], the Manhattan distance [5, 42], and the Euclidean distance [42]. In particular, the LSH family for the ℓ_p distance is found based on the p-stable distribution [42].

Definition 2.6 (p-stable distribution)**.** *A distribution G over $\mathbb{R}$ is called* p-stable*, if there exists $p \geqslant 0$ such that for any n real numbers $v_1, ..., v_n$ and i.i.d. random variables $X_1, ..., X_n$ with distribution G, the variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution G. It has been proved that stable distributions exist for $p \in (0, 2]$ [42]. In particular,*

- *The Cauchy distribution, with a density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable;*
- *The Gaussian distribution, with a density function $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.*

Using the p-stable distribution, one can generate a d-dimensional vector $\vec{a}$ by setting its element as a random value from the p-stable distribution. Given two vectors $\vec{v_1}, \vec{v_2} \in \mathbb{R}^d$, $(\vec{a}.\vec{v_1} - \vec{a}.\vec{v_2})$ is distributed as $\ell_p(\vec{v_1}, \vec{v_2})X$, where X is a random variable with the same p-stable distribution. Based on the above observations, Datar

et al. [42] proposed the following LSH family for the ℓ_p distance. The basic idea of LSH is that an LSH function produces a hash bit of a data point by projecting the data point to a random hyperplane. A data point in $\mathbb{R}^d$ is projected onto a random line $\vec{a}$, which is segmented into equi-width intervals with length r_0. Formally, the proposed LSH function in the ℓ_p space is defined as:

$$h(\vec{v}) = \lfloor \frac{\vec{a}.\vec{v} + b}{r_0} \rfloor, \tag{2.2}$$

where the projection vector $\vec{a} \in \mathbb{R}^d$ is constructed by picking each coordinate from a p-stable distribution.

Given $\vec{v_1}, \vec{v_2} \in \mathbb{R}^d$, let $s = \ell_p(\vec{v_1}, \vec{v_2})$. The probability that $\vec{v_1}$ and $\vec{v_2}$ collide under a hash function $h(\cdot)$, denoted as $p(s, r_0)$, can be computed as follows:

$$p(s, r_0) = \int_0^{r_0} \frac{1}{s} f_p(\frac{t}{s})(1 - \frac{t}{r_0})dt, \tag{2.3}$$

where $f_p(\cdot)$ is the probability density function of the absolute value of the p-stable distribution, and $p(s, r_0)$ is monotonically decreasing with s when r_0 is fixed [42]. As a result, the LSH family of the ℓ_p distance is $(1, c, p_1, p_2)$-sensitive with $p_1 = p(1, r_0)$ and $p_2 = p(c, r_0)$. For special cases such as p equals to 1 and 2, we can compute the probabilities using the corresponding density functions.

• For the Cauchy distribution ($p = 1$), we have

$$p(s, r_0) = 2\frac{\arctan(r_0/s)}{\pi} - \frac{1}{\pi(r_0/s)} \ln(1 + (r_0/s)^2) \tag{2.4}$$

• For the Gaussian distribution ($p = 2$), we have

$$p(s, r_0) = 1 - 2norm(-r_0/s) - \frac{2}{\sqrt{2\pi}(r_0/s)}(1 - e^{-(r^2/2s^2)}), \tag{2.5}$$

where $norm(\cdot)$ is the cumulative distribution function of the standard normal distribution.

The above LSH family was employed to process the approximate nearest neighbor queries in the ℓ_p space. Statistics guarantees that the nearby points in the projected hyperplane are likely to be close to each other in the original space. Next we generally introduce the LSH methods to answer the approximate nearest neighbor queries in the ℓ_p space and then discuss two most related methods to our approach: E2LSH and C2LSH.

Several approaches were proposed to answer approximate nearest neighbor queries in Euclidean space [42, 93, 132, 55, 129]. In those approaches, E2LSH [42] is the first LSH method that supports approximate nearest neighbor queries in the Euclidean space. The main drawback of E2LSH is that it needs to build multiple indexes with different radii, resulting in high maintenance overhead. To address the issue of high storage cost, multi-probe LSH [93] is proposed. It not only checks the data points that fall in the same bucket as the query point, but also probes multiple buckets that are likely to contain results in a hash table, which leads to the use of fewer hash tables. However, it still suffers from the need for building hash tables at different radii.

LSB-tree [132] is the first work that does not need to build hash tables at different radii. It groups hash values as a one-dimensional Z-order value and indexes it in a B-tree. Index entries for different radii can be retrieved in corresponding ranges in the B-tree. In addition, an LSB forest with multiple trees can be built to improve the search performance. C2LSH [55] further improves the LSB-tree method by introducing techniques of collision counting and virtual rehashing. It uses one hash function per hash table so that the number of hash buckets to read does not increase exponentially when the query radius increases. Furthermore, SRS [129] projects data points from the original high-dimensional space into a low-dimensional space via 2-stable projections. The major observation is that the ℓ_2 distance in the projected space over the ℓ_2 distance in the original space follows a chi-squared distribution, which has a

sharp concentration bound. Therefore, SRS can index the data points in the low-dimensional projected space and use the chi-squared distribution to perform queries. In this case, the index size is significantly reduced.

LSH variants are proposed to address the drawbacks of the traditional LSH methods. Traditional LSH methods suffer from the disadvantage of generating a large number of the false positives. To address this problem, BayesLSH [124] is proposed. It integrates the Bayesian statistics and performs candidate pruning and similarity estimation using LSH. It can quickly prune away false positives, which leads to a significant speedup. Traditional LSH methods also suffer from the disadvantage of accessing many candidates, which brings a larger number of random I/Os. To address this, SortingKeys-LSH [90] is presented to order the compound hash keys so that the data points are sorted accordingly in the index to reduce the I/O cost. Next we discuss two most related methods: E2LSH [42] and C2LSH [55].

E2LSH: E2LSH [42] is the first proposed LSH method to answer the $\mathcal{R}_p(\vec{q},\, r,\, c)$ problem in Euclidean space where $p = 2$, which is defined as: As can be seen, the $\mathcal{R}_p(\vec{q},\, r,\, c)$ problem is a decision version of the $\mathcal{N}_p(\vec{q}, k, c)$ problem. E2LSH exploits LSH functions to address the $\mathcal{R}_2(\vec{q},\, r,\, c)$ problem in the following way. First, a set of m LSH functions $h_1(\cdot), \ldots, h_m(\cdot)$ are randomly chosen from an $(r,\, cr,\, p_1,\, p_2)$-sensitive family $\mathcal{H}$, and they are concatenated to form a compound hash function $g(\cdot)$, where $g(\vec{p}) = (h_1(\vec{p}), \ldots, h_m(\vec{p}))$ for a point $\vec{p} \in \mathbb{R}^d$. By using a compound hash function instead of a single LSH function, the probability that two faraway points collide can be largely reduced. Then, the hash function $g(\cdot)$ is used to map all the data to a hash table. The above two steps are repeated for L times and accordingly, L compound hash functions $g_1(\cdot), \ldots, g_L(\cdot)$ are used to produce L hash tables.

When a query $\vec{q}$ comes, the points from buckets $g_1(\vec{q})$, ..., $g_L(\vec{q})$ are retrieved until all the points or the first $3L$ points are found. For each retrieved point $\vec{v}$, it is returned if $\vec{v} \in B_2(q, cr)$. A $\mathcal{N}_p(\vec{q}, k, c)$ query can be answered by issuing a series

of $\mathcal{R}_2(\vec{q},\, c,\, r)$ queries using gradually increasing search radii. For this purpose, hash tables with different radii must be built, incurring high storage cost.

C2LSH: To avoid building many hash tables for different radii, C2LSH [55] is proposed by changing the original hash function to:

$$h(\vec{v}) = \lfloor \frac{\vec{a}.\vec{v} + b^*}{r_0} \rfloor, \tag{2.6}$$

where b^* is uniformly drawn from $\lfloor 0, c^{\lceil \log_c td \rceil}(r_0)^2 \rfloor$, c is the approximation ratio, t is the largest coordinate value, and d is the dimensionality of the data. It is proved that the hash function is $(1,\, c,\, p_1,\, p_2)$-sensitive.

First, a materialized index is built for a set of base LSH functions with a small interval r_0. Then, C2LSH reuses the materialized index to retrieve objects at different radii without explicitly building hash tables for different radii. This process is referred to as *virtual rehashing*. To answer an $\mathcal{R}_2(\vec{q},\, r,\, c)$ query, C2LSH modifies the hash function as:

$$H^r(\vec{v}) = \lfloor \frac{h(\vec{v})}{r} \rfloor \tag{2.7}$$

Note that the $H^r(\vec{v})$ is $(r,\, cr,\, p_1,\, p_2)$-sensitive. *Virtual rehashing* simplifies the process of retrieving the objects hashed to $H^r(\vec{q})$ by guaranteeing that it is identical to retrieving objects in the buckets within $\left[\lfloor \frac{h_{\vec{a},b}(\vec{v})}{r} \rfloor \times r, \lfloor \frac{h_{\vec{a},b}(\vec{v})}{r} \rfloor \times r + r - 1\right]$ in the base hash function.

In addition, C2LSH estimates the probability of being the nearest neighbor using the *collision count*. If the number of an object colliding with a query exceeds a certain threshold, namely the *collision count threshold* θ, the object is likely to be a neighbor. Such an object is considered as a candidate and retrieved for computing its real distance to the query.

As a result, a $\mathcal{N}_p(\vec{q}, k, c)$ query can be answered by C2LSH by issuing a set of $\mathcal{R}_2(\vec{q},\, c,\, r)$ queries with increasing radii. C2LSH is claimed to be correct if these two

properties hold with a constant probability.

• $\mathcal{P}_1$: If $\vec{v} \in B_2(\vec{q}, r)$, then the number of $\vec{v}$'s collision with the query $\vec{q}$ is at least θ.

• $\mathcal{P}_2$: The total number of false positives is smaller than $\beta|\mathcal{D}|$, where $|\mathcal{D}|$ is the cardinality of the database $\mathcal{D}$.

In $\mathcal{P}_1$, the number of collisions θ is related to the number of base hash functions, which is denoted as η. Given an error probability ε and a false positive rate β, θ and η must be carefully tuned for the best performance.

Lemma 2.1. *If η and θ are set to:*

$$\eta = \lceil \frac{\ln \frac{1}{\varepsilon}}{2(p_1 - p_2)^2}(1+z)^2 \rceil, \text{ where } z = \sqrt{\frac{\ln \frac{2}{\beta}}{\ln \frac{1}{\varepsilon}}}, \tag{2.8}$$

$$\theta = \frac{zp_1 + p_2}{1+z}\eta, \tag{2.9}$$

then $Pr[\mathcal{P}_1] \geqslant 1 - \varepsilon$ and $Pr[\mathcal{P}_2] \geqslant 0.5$. [55]

Therefore, both $\mathcal{P}_1$ and $\mathcal{P}_2$ hold with a constant probability when the parameters are set as above, and C2LSH can correctly answer the $\mathcal{R}_2(\vec{q}, c, r)$ query.

2.3.3 Fractional Distance Metric

Euclidean distance [37, 64, 125, 15, 117] is widely used in kNN search. However, it was shown that when the dimensionality is high, the Euclidean distance introduces the concentration problem [17]. Namely, the distance between any random pair of high dimensional data points is almost identical. Therefore, the effectiveness of the Euclidean distance in the high-dimensional space is not clear [66, 2].

To address the concentration problem, one direction it to consider partial similarities, which have been drawn increasing attention in the last decade [82, 133, 23, 131]. Tung et al. [133] introduced the k-n-match model to discover the partial similarity in n dimensions, where n is a given integer smaller than dimensionality and these n

dimensions are determined dynamically to make the query and the returned results be the most similar. It has been shown that the k-n-match model yields better result than the traditional kNN query in identifying similar objects by partial similarities.

Another direction is to investigate different distance metrics. Aggarwal et al. [66, 2] examined the "curse of dimensionality" problem from the perspective of distance metrics. They found that the Manhattan distance (ℓ_1) metric is more effective than the Euclidean distance (ℓ_2) metric in the high-dimensional space. They further introduced and examined the fractional distance (ℓ_p for $0 < p < 1$) metrics, which are less concentrated. It was shown that the fractional distance metrics provide more meaningful results from both the theoretical and empirical perspectives. Their experimental results verified that fractional distance metrics improve the effectiveness of standard clustering algorithms.

Later, fractional distance metrics have been applied to applications such as content-based image retrieval [67, 136]. The experiments showed that the performances of fractional distance metrics outperform the ℓ_1 and ℓ_2 metrics. In particular, the $\ell_{0.5}$ distance consistently outperforms both ℓ_1 and ℓ_2 metrics in their experiments. Still, the experimental results showed that the optimal ℓ_p metric is application-dependent and needs to be learned for each dataset.

Recently, it was argued that the analysis of the concentration phenomenon is based on the assumption of independent and identically distributed variables [54], which might not be true in real datasets. The authors showed that the optimal ℓ_p metric is actually application-dependent. In order to explore the high-dimensional data, it is important to support different distance metrics. Hence, users can try and select the best ℓ_p metric for their applications. However, to the best of our knowledge, none of the existing LSH approaches can support multiple distance metrics. LazyLSH is the first work that supports approximate kNN processing in multiple fractional metrics using a single index, which will be presented in Chapter 4.

Chapter 3

INSPIRE: A Framework for Incremental Spatial Prefix Query Relaxation

Geo-textual data are generated in abundance. Recent studies focused on the processing of spatial keyword queries which retrieve objects that match certain keywords within a spatial region. To ensure effective retrieval, various extensions were done including the allowance of errors in keyword matching and autocompletion using prefix matching. In this chapter, we propose INSPIRE, a general framework, which adopts a unifying strategy for processing different variants of spatial keyword queries. We adopt the autocompletion paradigm that generates an initial query as a prefix matching query. If there are few matching results, other variants are performed as a form of relaxation that reuses the processing done in the earlier phase. The types of relaxation allowed include spatial region expansion and exact/approximate prefix/substring matching. Moreover, since the autocompletion paradigm allows appending characters after the initial query, we look at how query processing done for the initial query and relaxation can be reused in such instances. Compared to the existing work which process variants of spatial keyword query as new queries over different indexes, our approach offers a more compelling way to efficient and effective spatial keyword search. Extensive experiments substantiate our claims.

3.1 Motivation

As the growth in geographical applications like Google Earth and Foursquare, geo-textual data are generated in abundance. To retrieve such data effectively, recent studies focused on the processing of spatial keyword queries which retrieve objects that match certain keywords within a spatial region. In order to ensure effective and user-friendly retrieval, a popular paradigm is to support search-as-you-type as illustrated in Example 3.1.

Example 3.1. *In Figure 3.1a, a user wants to search for "Staples Center". He zooms in the region of Staples Center and types "staples ce". Object A is returned because it is in the viewport and starts with "staples ce" in its text.*

We call this query a Spatial Prefix (SP) query [13], and a point of interest (POI) is returned if it is located in the query region and the query text is a prefix of its textual content.

However, it is possible that no or few POIs are returned due to human error: (1) the query range (represented by the viewport) is wrongly specified by the user who is not familiar with the query region; (2) the query text does not satisfy the prefix condition due to typos in the query text or data uploaded by users. Query relaxation must be done to ensure that useful answers are returned. Such relaxation can be done in two ways: (1) expand the spatial region to a larger region; (2) relax the prefix condition to more general textual conditions. To relax the prefix condition, the first natural choice is to relax it to the *substring query*, which requires that the query text is an *exact substring* of the textual content of an object. A second type of relaxation is performed by allowing mismatches in keyword search but limiting the matching to start at the prefix. This is called *approximate prefix query*. If both of these types of relaxation fail to produce enough number of results, we would then

allow approximate string matching to be applied to any part of the textual content. This third type of relaxation is named *approximate substring query*.

Example 3.2. *Figure 3.1b shows a case of relaxing the spatial constraint. The considered spatial region is enlarged and object B is added to the result. Figures 3.1c-e show the cases of relaxing the prefix condition. In Figure 3.1c, the prefix constraint is relaxed to the substring matching and object C is added. In Figure 3.1d, the constraint is relaxed to the approximate prefix matching and object D is added. If at least three objects are required, the approximate substring matching is applied as in Figure 3.1e. Objects A, C, D and E are returned.*

As shown in Example 3.2, different queries may need different types and degrees of relaxation. Determining what relaxation to perform and performing each type of relaxation efficiently are major challenges. While existing methods handle some of these types of relaxation as a single query [13, 74, 140, 4], performing all these different types of relaxation means re-issuing a new query every time. This is inefficient especially when these queries are answered using different indexes in different work. Moreover, in the context of search-as-you-type, users can append characters to the initial query even as the initial query is being processed, giving rise to a new query which we refer to as an *appending query*.

Example 3.3. *In an interactive search, the user types two more characters "nt" to the query in Example 3.2. The text of the appending query becomes "staples cent". Instead of processing the appending query from scratch, we can use the results of the previous query and start processing it from the last type of relaxation, which is the approximate substring matching. Therefore, objects A, C, D and E are returned.*

Again, it is obvious that the appending query in the example can be handled by issuing it as a completely new query, but doing so will obviously be not efficient.

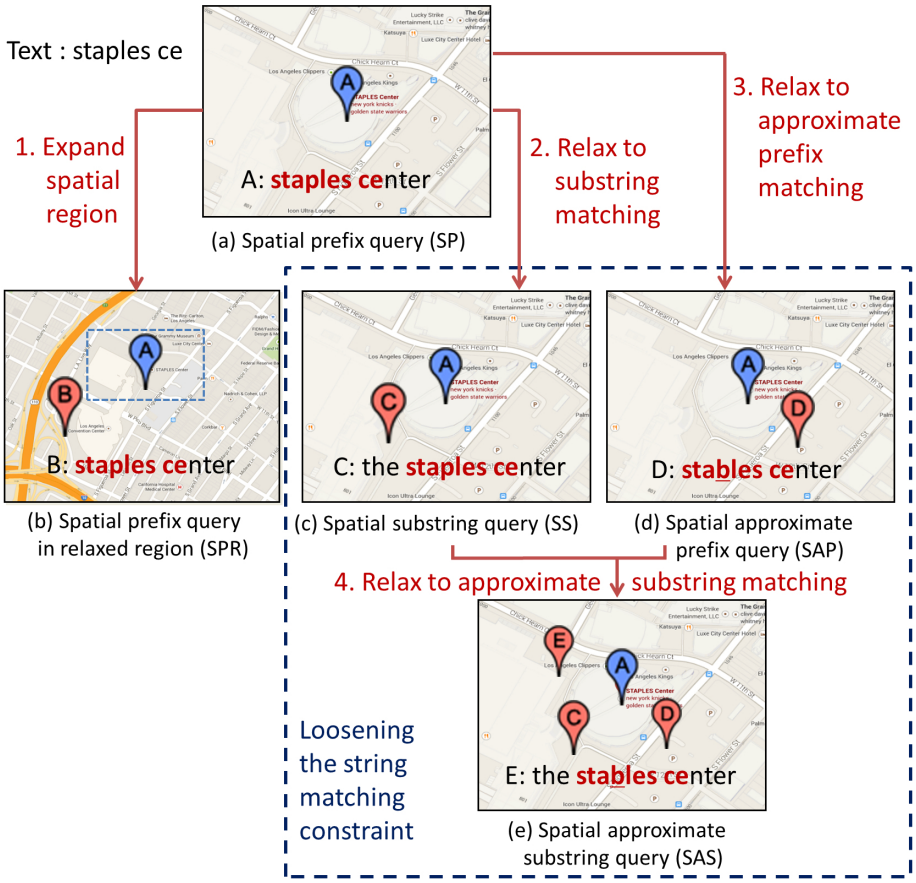


Figure 3.1: Spatial Prefix Query and Its Relaxation

In order to provide better solutions for the issues we discussed, we propose a general two-level inverted index that can support all types of relaxation that we have mentioned. The proposed structure has an object-level index to support functionalities and a node-level index to accelerate query processing. Using this two-level index, we propose several variants of the string filtering techniques to effectively remove data objects that do not satisfy the query condition.

Having observed that different types of relaxation have different levels of complexity and have dependencies among themselves, we propose INSPIRE, an INcremental Spatial PrefIx query RElaxation framework, to process the spatial prefix query and its relaxation. Similar to the strategy shown in Figure 3.1, the initial query is a spa-

tial prefix query. If no or few results are returned, we incrementally process different types of relaxation until sufficient results are returned. In particular, the spatial region of the prefix query is first expanded, followed by relaxation of different degrees on the string dimension: the substring query, the approximate prefix query and the approximate substring query.

By ordering these types of relaxation in an appropriate order, we can use the dependencies between them to optimize our query processing through the reuse of previous results. To further reduce the computational cost, we adapt selectivity estimation techniques to determine whether certain relaxation should be conducted. If the estimated selectivity does not meet a given threshold, we can simply go to the next-level relaxation. We will refer to the intermediate result reuse and selectivity estimation as *intra-query optimization*.

If a query is an appending query that is formed from a previous query by appending characters, we also look into how processing can be reused in such cases. In particular, we notice that a time-consuming process for answering a query is to merge inverted lists to obtain candidate results. We thus further optimize the processing of merging lists in appending queries. This is formalized as our *inter-query optimization*.

Our main contributions are summarized as follows:

1. We first identify the relationships among different types of relaxation for the spatial prefix query, and then propose a framework for incrementally processing the relaxation for a spatial prefix query (in Section 3.2).

2. We build a one-size-fits-all index (in Section 3.3) to support all types of relaxation (in Section 3.4).

3. During the incremental relaxation paradigm, we propose various result reuse methods to accelerate the processing of both non-appending and appending

queries, namely intra-query optimization (in Section 3.5) and inter-query optimization (in Section 3.6).

4. We conduct a comprehensive experimental study over three real data sets to demonstrate the efficiency and effectiveness of our methods (in Section 3.7).

3.2 The INSPIRE Framework

Our objective is to build a one-size-fits-all search engine for spatial prefix query and its various types of relaxation. In this section, we first introduce the preliminaries for such a search engine. Then we formally state our problem and introduce our INSPIRE framework.

3.2.1 Preliminary

Hilbert-encoded Quadtree: Quadtree [50] is widely adopted to facilitate fast retrieval of objects in multi-dimensional space. In a two-dimensional Quadtree, each node represents a bounding box covering a part of the space. The root node covers the whole space, and the leaf node contains indexed points, while the internal node has four children, one for each quadrant obtained by dividing the area covered in half along both axes.

A Hilbert curve [120] is a space-filling curve, which maps multi-dimensional data to one dimension. The Hilbert curve is defined recursively: each cell is divided into four subcells at each subsequent level. The sequential order of a cell in a curve is identified by its Hilbert code.

Definition 3.1 (Hilbert Code (hc))**.** *Given a Quadtree node n, its Hilbert code hc is a concatenation of the Hilbert code of its parent node and n's local order. Hilbert code hc_1 is defined to be ahead of hc_2, if hc_1 has a smaller value at the first different bit of hc_1 and hc_2.*

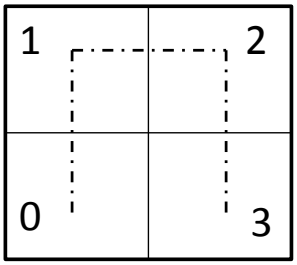


Figure 3.2: Hilbert Curve at Level One

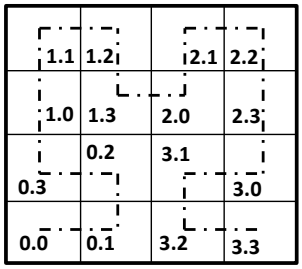


Figure 3.3: Hilbert Curve at Level Two

Example 3.4. *The Hilbert curve of level-two in Figure 3.3 is constructed from the Hilbert curve of level-one in Figure 3.2. When the local order is defined in range* $[0, 3]$*, the Hilbert codes of the nodes in the left-upper corner are 1.0, 1.1, 1.2 and 1.3 respectively. Node 1.0 is ahead of node 1.1.*

We integrate a Hilbert curve with a Quadtree as in [10]. As a result, we sequentialize the cells of a Quadtree to efficiently retrieve places located in a given spatial region. We call this a *Hilbert-encoded Quadtree*, on which we will build our novel index to handle the spatial prefix query and its various types of relaxation.

Example 3.5. *Figure 3.4 shows a Hilbert-encoded Quadtree of eleven objects. A Quadtree is first built. The maximum capacity of each leaf node is set to two. Then a Hilbert curve is built to link the nodes of the Quadtree, which is plotted as the dotted curve. Each leaf node is attached with a Hilbert code to represent its sequential order in the Hilbert curve.*

String Filtering: The approximate search is used to handle cases when queries have errors. Among the literature for handling approximate string query, most work builds solutions based on the concept of *q-grams* [60, 137, 87, 31].

Definition 3.2 ((Positional) q-gram)**.** *A q-gram tk is a sequence of q characters in string s. A positional q-gram (tk,p) is a pair of q-gram tk with its position p in string s.*

To support approximate string queries, Gravano et al. proposed an efficient solution for edit similarity join [60]. The edit distance constraint is relaxed to a weaker constraint based on the number of matching q-grams.

Definition 3.3 (Matching (Positional) q-gram)**.** *A q-gram tk_1 matches tk_2 if $tk_1 = tk_2$. Given an edit distance threshold τ, a positional q-gram (tk_1, p_1) matches (tk_2, p_2) if $tk_1 = tk_2$ and $|p_1 - p_2| \leqslant \tau$.*

However, one problem of [60] is that a string candidate pair cannot be discarded unless all their q-grams have been compared. To quickly filter out candidate pairs, a prefix-based filtering is proposed [137]. We summarize some widely adopted filters in the approximate string search as follows.

Definition 3.4 (Filters of String Approximation)**.** *Given strings s, t, and an edit distance threshold τ, the following filters are used to check whether the edit distance between s and t is within τ.*

*(1) Length Filtering (**LF**) [60]: $||s| - |t|| \leqslant \tau$.*

*(2) Count Filtering (**CF**) [60]: s and t must share at least $LB_{s,t,\tau,q}$ = (max($|s|$, $|t|$) - q + 1) - q * τ matching q-grams.*

*(3) Position Filtering (**PoF**) [60]: s and t must share at least $LB_{s,t,\tau,q}$ matching positional q-grams.*

*(4) Prefix Filtering (**PrF**) [137]: the first (q * τ + 1) positional q-grams of s and t must have at least one match.*

3.2.2 Problem Statement

The problem we are going to address is to efficiently answer the spatial prefix query and its various types of relaxation. It can be reduced to two problems: (1) To design an incremental relaxation paradigm, and the relaxation is triggered if no or few results are returned. (2) To provide the search-as-you-type feature for the interactive search.

Table 3.1: Table of Notations for Chapter 3

n	a node of a Quadtree	O	a spatial object
$n.hc$	the hilbert code of n	$p\top$	reserved position
θ	result size threshold	τ	edit distance threshold
ω	string query type	$\Re$	prefix ratio
Q'_ω	appending query	Q_ω	previously-formed query
$Res(Q_\omega)$	result set of Q_ω	ϵ	scarce threshold
$S(Q_\omega)$	estimated selectivity of Q_ω		
$S_\omega(n, s)$	estimated selectivity of string s in n for string type ω		
q_c (q_p)	q value for q-gram (positional q-gram)		
$NS(n, Q_\omega)$	node snippet of node n with respect to Q_ω		
$LB_{s,\tau,q}$	min match threshold, ($\|s\|$ - q + 1) - q * τ		
$UB_{p^\top,\tau,q}$	length bound, ($p\top$ - τ + q - 1)		

The frequently used notations are summarized in Table 4.2. Each spatial object O is defined as a tuple ($O.id$, $O.loc$, $O.str$), where $O.id$ is an identification, $O.loc = (lat, lng)$ the latitude and longitude, and $O.str$ the textual content. A spatial prefix query is defined as:

Definition 3.5 (Spatial Prefix Query)**.** *A Spatial Prefix (SP) query Q_p = (rng, str) consists of two parts: (1) the spatial region $Q_p.rng$, which can be retrieved from the user's viewport; (2) the query string $Q_p.str$.*

A spatial object O is an answer to a SP query Q_p if: (1) $O.loc$ is in $Q_p.rng$, and (2) $Q_p.str$ is a prefix of $O.str$. When a spatial prefix search returns no or few results, we relax the query in order to obtain sufficient results. The spatial prefix query can be relaxed in two ways: (1) relax the spatial constraint, or (2) relax the prefix constraint.

Definition 3.6 (Relaxation of Spatial Region)**.** *Given an SP query Q_p = (rng, str), the relaxation on the spatial region leads to a Spatial Prefix query in a Relaxed region (SPR) Q^r_p = (rng′, str), where $Q_p.rng \subset Q^r_p.rng'$.*

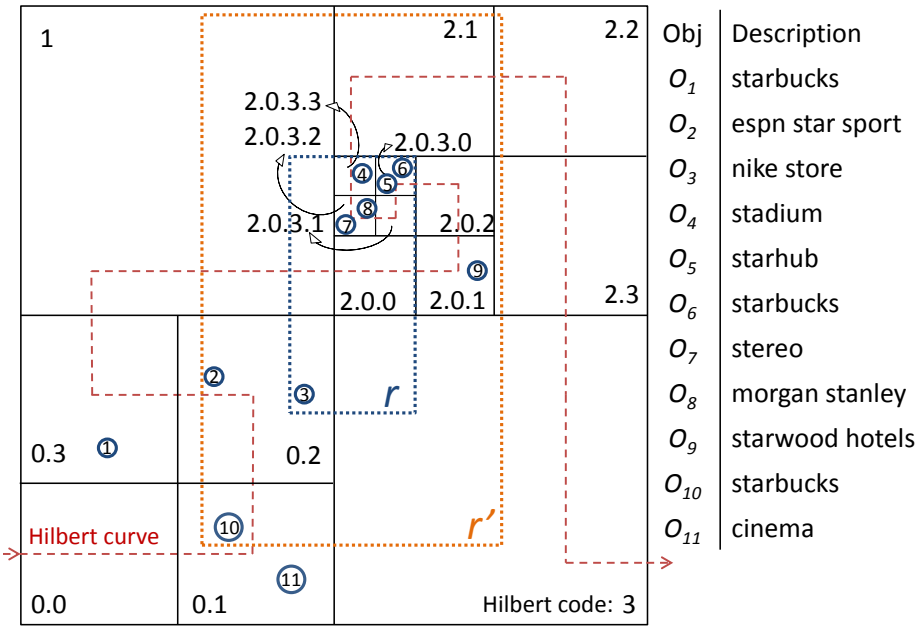


Figure 3.4: Hilbert-encoded Quadtree of Eleven Objects

Definition 3.7 (Relaxation of Prefix Constraint)**.** *Given an SP query Q_p = (rng, str), the relaxation on the prefix constraint leads to a relaxed query Q_ω = (rng, str, (τ)), where ω is one of the following string queries: exact substring query (s), approximate prefix query (ap) [31] and approximate substring query (as) [59]. τ is an edit distance threshold for the latter two types of approximations. Therefore, the three types of relaxation of the prefix constraint are:*

(1) Spatial Substring (SS) query: Q_s.

(2) Spatial Approximate Prefix (SAP) query: Q_{ap}.

(3) Spatial Approximate Substring (SAS) query: Q_{as}

Example 3.6. *Figure 3.4 shows a spatial database. The textual contents of the objects are shown on the right, while the locations are shown on the left. Table 3.2 illustrates an example of an SP query, Q_p = (r, "star"), and its relaxation.*

As reported in [88], users usually input a query letter by letter, and newly typed characters are appended to the previous query forming a new query at any moment.

Table 3.2: Spatial Prefix Query and its Relaxation

Query type	Notation	Query Result
SP	$Q_p = (r,$ "star")	O_5, O_6
SPR	$Q_p^r = (r',$ "star")	O_5, O_6, O_9, O_{10}
SS	$Q_s = (r,$ "star")	O_5, O_6
SAP	$Q_{ap} = (r,$ "star", 1)	O_4, O_5, O_6, O_7
SAS	$Q_{as} = (r,$ "star", 1)	$O_3, O_4, O_5, O_6, O_7, O_8$

We call this type of query an *appending query*. Processing of appending queries is important to achieve the search-as-you-type paradigm for the interactive search.

Definition 3.8 (Appending Query Q'_ω)**.** *Given two queries Q_ω and Q'_ω, Q'_ω is an appending query of Q_ω if (1) $Q_\omega.rng$=$Q'_\omega.rng$, and (2) $Q_\omega.str$ is a prefix of $Q'_\omega.str$.*

Example 3.7. *In the interactive search, the user then issues another query Q'_p = (r, "starbu"), which is an appending query of Q_p = (r, "star"). The answer to Q'_p is O_6.*

3.2.3 Framework

We use $Res(Q_\omega)$ to denote the result set of a spatial query Q_ω. For a spatial prefix query and its various types of relaxation, we identify the following relationships:

Observation 3.1 (Inter-relaxation Relationships)**.**

(1) $Res(Q_p) \subseteq Res(Q_p^r)$,

(2) $Res(Q_p) \subseteq Res(Q_s) \subseteq Res(Q_{as})$,

(3) $Res(Q_p) \subseteq Res(Q_{ap}) \subseteq Res(Q_{as})$

Besides, we observe the *inter-query relationship* between a previously-formed query and its appending query. Further details will be presented in Section 3.6.1.

Observation 3.2 (Inter-query Relationships)**.**

(1) $Res(Q_p) \supseteq Res(Q'_p)$, *(2)* $Res(Q_p^r) \supseteq Res(Q_p'^r)$,

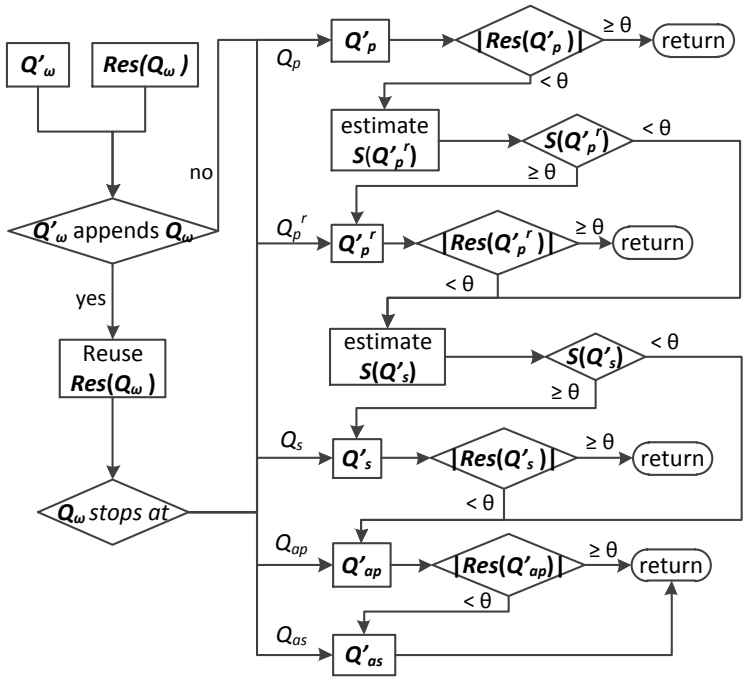


Figure 3.5: The INSPIRE Framework

(3) $Res(Q_s) \supseteq Res(Q'_s)$*, (4)* $Res(Q_{ap}) \supseteq Res(Q'_{ap})$*,*

(5) $Res(Q_{as}) \supseteq Res(Q'_{as})$

The above two containment relationships motivate us to propose an incremental relaxation strategy, called INcremental Spatial PrefIx query RElaxation (INSPIRE). As shown in Figure 3.5, the INSPIRE framework processes the spatial prefix query in the following way. Given an incoming query Q'_ω, we first check whether Q'_ω is an appending query of another query Q_ω that has been processed previously.

Case 1: If Q'_ω is not an appending query, we process it as a new SP query. If the result size of Q'_p is not less than a certain threshold θ, the results are returned. Otherwise, the relaxation is applied incrementally: the SPR query is first processed, followed by the SS, SAP and SAS queries.

Case 2: If Q'_ω is an appending query of Q_ω, we process it from where Q_ω stops. We first reuse $Res(Q_\omega)$ to form the candidate results of Q'_ω. If sufficient results are

found, processing for Q'_ω stops; otherwise, we further relax Q'_ω in an incremental way as what we have done in Case 1.

In Case 1, the incremental relaxation for a non-appending query may contain several phases, and Observation 3.1 identifies the commonality among relaxation at different levels. It motivates us to reuse the results of the less relaxed queries (computed in earlier phases) to process the query in the new phase. Furthermore, we propose a method to estimate the selectivity of a query, denoted as $S(Q'_\omega)$, ahead of processing it; as a result, if $S(Q'_\omega)$ is smaller than the threshold, we can skip the relaxation and go to the relaxation at the next level. The reuse and estimation form the *intra-query optimization*, which will be presented in Section 3.5.

In Case 2, Observation 3.2 brings two optimization opportunities: (1) We can reuse the results of previously-formed query Q_ω w.r.t. the appending query Q'_ω. (2) We do not process Q'_ω from scratch; instead, we process Q'_ω from where Q_ω stops. They are formalized as our *inter-query optimization*, which will be presented in Section 3.6.

3.3 Two-level Inverted Index

To support INSPIRE, we need a one-size-fits-all index that can answer the spatial prefix query and support the computation of its different types of relaxation. We propose an index structure comprising two parts: a *Hilbert-encoded Quadtree* in the main memory, which is described in Section 3.2.1, and a two-level inverted index on the disk. We introduce such an index in this section.

3.3.1 Spatial q-gram

As shown in Section 3.2.1, building an inverted index on q-grams is a natural choice in string search. To answer a spatial prefix query and its relaxation, we propose a two-level inverted index, which is essentially an adaption of the inverted index from

the granularity of q-gram to a finer level of granularity called *spatial q-gram*.

Definition 3.9 (Spatial (Positional) q-gram)**.** *A spatial q-gram ζ = (hc, tk) contains a Hilbert code hc and a q-gram tk. A spatial positional q-gram ξ = (hc, tk, p) consists of a Hilbert code hc and a positional q-gram (tk, p).*

The spatial q-gram links the Hilbert code and the q-gram to capture the spatial and textual information of an object. By extending the concept of *matching q-gram* in Definition 3.3, the matching spatial q-gram is defined and will be used to prune objects that cannot satisfy the textual constraint (see Section 3.4.1).

Definition 3.10 (Matching Spatial q-gram)**.** *Spatial q-grams ζ_1 and ζ_2 match if $\zeta_1.hc = \zeta_2.hc$ and $\zeta_1.tk = \zeta_2.tk$. Spatial positional q-grams ξ_1 and ξ_2 match w.r.t. a threshold τ, if $\xi_1.hc = \xi_2.hc$, $\xi_1.tk = \xi_2.tk$ and $|\xi_1.p - \xi_2.p| \leqslant \tau$.*

3.3.2 Object-level & Node-level Inverted Index

Once the *Hilbert-encoded Quadtree* is constructed, we build a two-level inverted index. To obtain the spatial and textual information of an object, we build an object-level inverted index on spatial q-grams by setting a spatial q-gram as the key and the objects containing the spatial q-gram as the value.

Besides the object-level inverted index, we need to know the number of objects having a particular q-gram in a node to estimate the selectivity to optimize the query processing. Thus, we attach a count to each leaf node of the Quadtree to record the number of objects containing a particular q-gram in a node. By setting a q-gram as the key and a list of (Hilbert-code, count) pairs as the value, we build a coarse-grained index, called node-level inverted index. We will present the selectivity estimation later in Section 3.5.1.

Example 3.8. *Table 3.3a illustrates the object-level inverted index of the database in Figure 3.4. The first entry, (0.2,"sta") $\rightarrow \{O_2\}$, indicates that O_2 is located in*

Table 3.3: Two-level Inverted Index

(a) Object-level Inverted Index

Spatial 3-gram	IDs
(0.2, "sta")	O_2
(0.2, "tar")	O_2
(2.0.3.0, "sta")	O_5, O_6
(2.0.3.0, "tar")	O_5, O_6
.....	.....

Spatial positional 2-gram	IDs
(0.2, "ar", 7)	O_2
(0.2, "st", 5)	O_2, O_3
(0.2, "ta", 6)	O_2
(2.0.3.0, "ar", 2)	O_5, O_6
(2.0.3.0, "st", 0)	O_5, O_6
(2.0.3.0, "ta", 1)	O_5, O_6
.....	.....

(b) Node-level Inverted Index

3-gram	(Hilbert code, count) list
"sta"	(0.1, 1), (0.2, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2),(2.0.3.2, 1), (2.0.3.3, 1)
"tar"	(0.1, 1), (0.2, 1), (0.3, 1), (2.0.1, 1),(2.0.3.0, 2)
.....	.....

positional 2-gram	(Hilbert code, count) list
("ar", 2)	(0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2)
("ar", 7)	(0.2, 1)
("st", 0)	(0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2), (2.0.3.2, 1), (2.0.3.3, 1)
("st", 5)	(0.2, 2)
("st", 7)	(2.0.3.2, 1)
("ta", 1)	(0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2), (2.0.3.3, 1)
("ta", 6)	(0.2, 1)
.....	.....

the leaf node with Hilbert code "0.2" and has a 3*-gram "sta". Table 3.3b shows the node-level inverted index. From the first entry "sta"* $\rightarrow$ {*(0.1, 1), . . . , (2.0.3.3, 1)*}*, we know that there is one object located in node "0.1" and containing a* 3*-gram "sta", etc.*

B$^+$ trees are built on the keys in the inverted index for quick access to index entries. The keys of the inverted index are sorted. For example, the spatial positional q-gram is sorted by the Hilbert code, then by the alphabetical order of the q-gram and the q-gram position. The values of the node-level and object-level inverted indexes are sorted by the Hilbert code of a node and the object ID respectively to support the filters in Section 3.4.1.

3.3.3 Combining q-gram with positional q-grams

Positional q-gram is proposed to accelerate the processing of SP, SPR and SAP queries at the expense of large storage space [60, 137, 146]. However, as reported in [127], the average number of words per query is small (2.35) and the number of words per query is smaller than three in most cases (87.4%). Thus, it is not worthwhile to store all positional q-grams. Instead, we only store the positional q-grams with position less than a certain value, referred to as *reserved position* $p^{\top}$. The setting of $p^{\top}$ will be presented in the experiment.

A compromise between storage and efficiency is made for storing less positional q-grams. However, it leads to a problem that SS and SAS queries cannot be supported. As compensation, we also store q-grams (NOT positional). Compared to the large storage required for positional q-grams, the storage of q-grams is much smaller. Therefore, we can store q-grams with larger q value to achieve better performance. For ease of illustration, we use (q_p) q_c to denote the q value for (positional) q-grams . As a result, inverted indexes based on positional q_p-grams and q_c-grams are built as shown in Table 3.3.

3.3.4 Index Implementation & Maintenance

Index Implementation: Our index structure consists of two parts: the Hilbert-encoded Quadtree in the main memory and the two-level inverted index on the disk. B^+ trees are built on the keys in our two-level inverted index for quick access to index entries. Two B^+ trees are used to store the object-level inverted index (for spatial q_c-grams and spatial positional q_p-grams respectively), and two B^+ trees to store the node-level inverted index (for q_c-grams and positional q_p-grams respectively).

The keys are ordered to fast retrieve index entries.

- The q_c-gram is ordered by the alphabetical order of the q_c-gram.

- The positional q_p-gram is first ordered by alphabetical order of the q_p-gram and then by the position.

- The spatial q_c-gram is first ordered by the sequential order of the hilbert code, then by the alphabetical order of the q_c-gram.

- The spatial positional q_p-gram is first ordered by the sequential order of the hilbert code, then by the alphabetical order of the q_p-gram and last by the position.

In addition, the value parts are also ordered to facilitate the processing of the node-level and object-level filters described in Section 3.4.1.

- For the node-level inverted index, the value is ordered by the object ID.

- For the object-level inverted index, the value is ordered by the sequential order of the hilbert code of the node.

Index Maintenance: The construction and maintenance of our proposed index contain three parts: insertion, deletion and update. The details are listed as follows.

Insertion: Inserting a new spatial object contains two parts. One is inserting the object to the Hilbert-encoded Quadtree, which completes in two steps: (1) inserting the object to the Quadtree and (2) updating the Hilbert code if a node is split.

The other is updating the two-level inverted index. For the object-level inverted index, we retrieve the inverted lists for the spatial (positional) q-grams of the new object and insert the object id to the lists. For the node-level inverted index, we get the inverted lists for the (positional) q-grams of the object and add one to the count of the node where the object is located.

When a node overflows, it is split into four new nodes. First we delete the inverted entries for the parent node. Then we update the statistics of (positional) q-grams

for the new nodes and the information of spatial (positional) q-grams for the objects in the new nodes.

Deletion: When deleting an existing object, the inverse operations are performed. Deletion also contains two parts. One is deleting the object from the Hilbert-encoded Quadtree, which completes in two steps: (1) deleting the object from the Quadtree and (2) updating the Hilbert code if we need to collapse the sibling nodes and merge them into one.

The other is updating the two-level inverted index. For the object-level inverted index, we retrieve the inverted lists for the spatial (positional) q-grams of the object and delete the object id in the lists. For the node-level inverted index, we get the inverted lists for the (positional) q-grams of the object and subtract one from the count of the node where the object is located.

When we delete an object from a leaf node, we get the number of remaining objects in the leaf and its sibling nodes. If the number is smaller than the max capacity of the Quadtree, we need to merge the nodes into one. First we delete the inverted entries for the leaf node and its siblings. Then we update the statistics of (positional) q-grams for the merged node and the information of spatial (positional) q-grams for the objects in the merged node.

Update: When updating an existing spatial object, we first check whether the object moves to a different leaf node of the Hilbert-encoded Quadtree. If so, we treat this update as a deletion of the existing object and a new insertion of the updated object. Otherwise, we only need to update the two-level inverted index correspondingly.

3.3.5 Space Requirement

Given that there are u unique q-grams in a database and a text has v q-grams, Navarro and Baeza-Yates[105] assumed that q-grams are uniformly distributed and

proved that the average number of unique q-grams in the text is $\Theta(\min(v, u))$. We analyse the required index space based on it.

We assume that the text of an object has v q-grams on average. After inserting n spatial objects in the database to the *Hilbert-encoded Quadtree*, it ends up with m leaf nodes. The number of leaf nodes is deeply affected by the fanout of a node, which is an important parameter to be discussed in Section 3.7. Each leaf node contains (n/m) spatial objects and (nv/m) q-grams on average. Based on [105], each node contains $\Theta(\min(nv/m, u))$ unique q-grams on average. Thus, there are $(m \cdot \Theta(\min(nv/m, u)))$ inverted lists in the object-level inverted index on average, and the average length of these inverted lists is $\Theta(\max(nv/mu, 1))$. As a result, the space requirement for the object-level inverted index is $\mathrm{O}\,(nv)$.

For the node-level inverted index, the portion of unique q-grams that a leaf node contains is $\min(nv/mu, 1)$. Consequently, the average length of these inverted lists is $\Theta(\min\ (nv/u,\ m))$. Since there are u inverted lists, the space requirement for the node-level inverted index is $\mathrm{O}\,(mu)$. We will show the empirical sizes in the experimental section.

3.4 Query Processing

In INSPIRE, the SP query is first processed. If its result size is smaller than the result size threshold θ, different types of relaxation are applied incrementally. In this section, we present how a single query is processed. For processing an appending query, we will present it in Section 3.6.

As shown in Algorithm 1, processing a query Q_ω has two key steps: (1) get the candidate nodes (lines 1-2), and (2) get the results for each candidate node (lines 6-10).

First, we present how to get the candidate nodes. We get the leaf nodes of the

Algorithm 1: Answering Q_ω

```
1  Intr(Q_ω) ← getIntrNodeSnippet(Q_ω) ;
2  FtNode ← nodeLevelFilter(Q_ω) ;
3  CanNode ← Intr(Q_ω) ∩ FtNode ;
4  foreach n ∈ CanNode do
5      CanObj ← Obj(n, Q_ω) ;
6      FtObj ← objectLevelFilter(Q_ω, n) ;
7      CanObj ← CanObj ∩ FtObj ;
8      foreach o ∈ CanObj do
9          if verifyObject(o) then
10             Result ← Result ∪ o ;
11 return Result ;
```

Quadtree that intersect with $Q_\omega.rng$, denoted as $Intr(Q_\omega)$ (line 1). Then the *node-level filter* takes $Intr(Q_\omega)$ as the input and removes the nodes that cannot satisfy the string constraint (line 2). For each node $n \in Intr(Q_\omega)$, its snippet is retrieved.

Definition 3.11 (Node Snippet)**.** *The node snippet of a node n with respect to a query Q_ω, denoted as $NS(n, Q_\omega)$, contains: (1) n.hc: the Hilbert code of node n.*
(2) $Obj(n)$: the objects located in node n.
(3) $Obj(n, Q_\omega)$: the objects located in node n and $Q_\omega.rng$. $Obj(n)$ and $Obj(n, Q_\omega)$ are used to estimate the selectivity of a query, which will be illustrated in Section 3.5.1.

Then we show how to get the result objects for each candidate node. For a candidate node n, $Obj(n, Q_\omega)$ is retrieved when getting the snippet of n. The *object-level filter* of node n takes $Obj(n, Q_\omega)$ as the input and prunes the objects that cannot meet the string constraint (line 6). Finally, the candidate objects are verified whether they truly satisfy the textual constraint (lines 8-10).

3.4.1 Node-level and Object-level Filters

Filters are used at two levels: *nodeLevelFilter*() and *objectLevelFilter*() to filter nodes and objects that cannot satisfy the string constraint respectively. Recall Definition 3.4, string filters have been proposed in the literature for approximate queries. To cater for the processing of query Q_ω based on our two-level inverted index, we propose variants of them at the node level and object level respectively. Given a query text s and an edit distance threshold τ, the node-level and object-level filters are:

Definition 3.12 (Node-level Filters)**.**

CF$_n$*: a node n must have at least* $LB_{s,\tau,q_c} = (|s|\text{-}q_c\text{+}1)\text{-}q_c \cdot \tau$ *matching* q_c*-grams.*

PoF$_n$*: a node n must have at least* LB_{s,τ,q_p} *matching positional* q_p*-grams.*

PrF$_n$*: a node n must have at least one matching positional* q_p*-gram in the first* (q_p τ τ+1) *positional* q_p*-grams of s.*

Definition 3.13 (Object-level Filters in Node n)**.**

CF$_o$*: an object o must have at least* LB_{s,τ,q_c} *matching spatial* q_c*-grams in node n.*

PoF$_o$*: an object o must have at least* LB_{s,τ,q_p} *matching spatial positional* q_p*-grams in node n.*

PrF$_o$*: an object o must have at least one matching spatial positional* q_p*-gram in the first* (q_p τ τ+1) *spatial positional* q_p*-grams of s in node n.*

Table 3.4 shows the filters used for each type of query. Column 1 is the query type, columns 3 and 2 specify the filters adopted and the condition to trigger such filters, while column 4 specifies the minimum match threshold for a qualified candidate to pass the filter.

In particular, PoF can only be applied when the query length is not greater than $UB_{p^\top,\tau,q_p}$, where $UB_{p^\top,\tau,q_p} = (p^\top \text{ - } \tau + q_p \text{ - } 1)$. Similarly, PrF can be applied when τ is not greater than $(p^\top/q)$ in SAP queries. Otherwise, we cannot get all the matching

Table 3.4: Filters used in a particular query

Query	Condition	Filter	min match
SP & SPR	$\|s\| \leqslant UB_{p^\top,0,q_p}$	PoF	$LB_{s,0,q_p}$
	$\|s\| > UB_{p^\top,0,q_p}$	CF	$LB_{s,0,q_c}$
SS		CF	$LB_{s,0,q_c}$
SAP	$\tau \leqslant p\top/q_p$	PrF	1
	$\|s\| \leqslant UB_{p^\top,\tau,q_p}$	PoF	LB_{s,τ,q_p}
	$\|s\| > UB_{p^\top,\tau,q_p}$	CF	LB_{s,τ,q_c}
SAS		CF	LB_{s,τ,q_c}

positional q_p-grams from the proposed index because we only store the positional q_p-grams with position less than $p^\top$ (recall Section 3.3.3).

The essence of the node-level and object-level filters is to obtain the candidate nodes and objects that appear at least a certain number of times on the given inverted lists. It is formalized as the *list merging problem* in [122, 87]. The minimum number of appearances corresponds to the min match threshold in the 4^{th} column of Table 3.4.

For the node-level filters, the lists are retrieved from the node-level inverted index using the matching (positional) q-grams. For the object-level filters, the lists are retrieved from the object-level inverted index using the matching spatial (positional) q-grams constructed from the candidate node and the query text.

Here, we present how a query is processed using an example. We show how to answer an SAP query, Q_{ap} = (r, "star", 1), where $\tau = 1$ and $Q_{ap}.r$ is shown in Fig. 3.4.

Example 3.9. *First, we retrieve the intersecting leaves $Intr(Q_{ap})$: {0.2, 1, 2.0.0, 2.0.3.0, 2.0.3.1, 2.0.3.2, 2.0.3.3}.*

Second, we use the node-level filters to prune the nodes that cannot satisfy the string constraint. Here, we only describe PoF_n. PoF_n with minimum match $LB_{s,\tau,q_p} =$ 1 is used. The matching positional 2-grams for $Q_{ap}.str$="star" are ("st", 0), ("st", 1), ("ta", 0), ("ta", 1), ("ta", 2), ("ar", 1), ("ar", 2), ("ar", 3). The inverted

lists for the matching positional 2-grams are retrieved and the intersection operation is performed between each inverted list and $Intr(Q_{ap})$. The nodes appearing at least $LB_{s,\tau,q_p} = 1$ time in the inverted entries are candidates. By employing the list merging algorithms [87], we get the candidate nodes, CanNode: {2.0.3.0, 2.0.3.2, 2.0.3.3}.

Third, we apply the object-level filters to prune objects in each candidate node. The procedure is similar to that of node-level filters. The major difference is that we retrieve objects from the object-level inverted index using matching spatial positional q-grams. Take node 2.0.3.0 as an example. The Hilbert code (2.0.3.0) and the matching positional 2-grams form the matching spatial positional 2-grams (recall Definition 3.10). By employing the object-level filters in node 2.0.3.0, we obtain the candidate objects: {O_5, O_6}.

Finally, O_5 and O_6 are checked by the query constraints. They are returned as answers.

3.4.2 Complexity Analysis

As shown in Algorithm 1, the node-level filters are first applied to obtain candidate nodes. For each candidate node, the object-level filters are applied to get candidate objects. We adopt the MergeSkip algorithm [87] for these filters to get candidates, which takes $\mathrm{O}(wL)$ time, where w is the number of merging lists and L is the average length of these lists. Suppose a query contains w q-grams, there are w lists to be merged. As stated in Section 3.3.5, $L = \mathrm{O}(m)$ for the node-level inverted index and $L = \mathrm{O}(nv/mu)$ for the object-level inverted index. Suppose that the number of candidate nodes is c, the object-level filters are applied c times. Therefore, Algorithm 1 can be processed in $\mathrm{O}(wm + cwnv/mu)$.

3.5 Intra-Query Optimization

Recall the framework in Section 3.2.3, since we adopt an incremental relaxation, it will be beneficial if we could skip processing a certain type of relaxation, whose result size does not exceed the result size threshold. It motivates our first optimization: estimating the selectivity of a query before its processing is triggered. In addition, we can use the dependencies between the relaxation at different levels to optimize our query processing through reuse of intermediate results. These two form intra-query optimization. In this section, we present how to exploit such intra-query optimization to accelerate the processing of the incremental relaxation.

3.5.1 Selectivity Estimation

Given a spatial query Q_ω=(rng,str,τ), we compute its selectivity $S(Q_\omega)$ in a bottom-up manner. We first retrieve the set of leaf nodes that intersect with $Q_\omega.rng$, denoted as $Intr(Q_\omega)$. For each node $n \in Intr(Q_\omega)$, we estimate the selectivity for string $Q_\omega.str$ of query type ω in node n, denoted as $S_\omega(n, Q_\omega.str)$. As we calculate $S(Q_\omega)$, we need to multiply $S_\omega(n, Q_\omega.str)$ by the proportion of node n inside $Q_\omega.rng$. Similar to [1], we assume that objects are uniformly distributed inside each node. Thus this proportion is calculated as $|Obj(n, Q_\omega)|/|Obj(n)|$. Finally, we sum up the calculated values to obtain $S(Q_\omega)$:

$$S(Q_\omega) = \sum_{n \in Intr(Q_\omega)} \frac{|Obj(n, Q_\omega)|}{|Obj(n)|} S_\omega(n, Q_\omega.str), \tag{3.1}$$

where $Intr(Q_\omega)$, $|Obj(n, Q_\omega)|$ and $|Obj(n)|$ can be retrieved from *getIntrNodeSnippet*(Q_ω) (line 1 in Algorithm 1). Next we show how to compute $S_\omega(n, Q_\omega.str)$.

3.5.1.1 Baseline method: Markov Estimator

The Markov Estimator (ME) is used to estimate the substring selectivity in relational databases [70, 30]. It is based on the Markov chain assumption, which states that "the probability of observing a q-gram in a sequence depends only on the q-gram immediately preceding it, and is independent of all other preceding q-grams." We apply the Markov Estimator to estimate the *substring selectivity* of a node. For example, the *substring selectivity* of "star" in node n is:

$$\begin{aligned} ME_s(n, \text{“}star\text{”}) &= P(\text{“}star\text{”}|\text{“}sta\text{”}) \cdot P(\text{“}sta\text{”}) \cdot |n| \\ &\approx P(\text{“}tar\text{”}|\text{“}ta\text{”}) \cdot P(\text{“}sta\text{”}) \cdot |n| \\ &= C_n(\text{“}sta\text{”}) \cdot C_n(\text{“}tar\text{”})/C_n(\text{“}ta\text{”}), \end{aligned} \tag{3.2}$$

where $C_n(tk)$ is the number of objects containing the q-gram tk in node n, and can be retrieved from the node-level inverted index using tk as the key. The same example can be used to present the *prefix selectivity* estimation of a node.

$$\begin{aligned} ME_p(n, \text{“}star\text{”}) &= P((\text{“}star\text{”}, 0)|\text{“}star\text{”}) \cdot P(\text{“}star\text{”}) \cdot |n| \\ &\approx (C_n(\text{“}st\text{”}, 0)/C_n(\text{“}st\text{”})) \cdot ME_s(n, \text{“}star\text{”}), \end{aligned} \tag{3.3}$$

where $C_n(tk, p)$ is the number of objects containing the positional q-gram (tk, p) in n, and can be retrieved from the node-level inverted index using (tk, p) as the key. The term $(C_n(tk, p)\ /C_n(tk))$ is denoted as the prefix ratio $\Re$.

However, Chaudhuri et al. [30] discovered that the Markov Estimator underestimates the true selectivity when a query contains a short identifying substring.

Definition 3.14 (Short Identifying Substring [30])**.** *A substring t' of t is a short identifying substring if*

(1) the selectivity of t' is close to the selectivity of t, and

(2) t' is much shorter than t.

3.5.1.2 Scarce q-gram method

As shown in [30], if string t contains a short identifying string t', the estimated selectivity of t' is much closer to the true selectivity of t. It motivates us to define *scarce q-gram* at the node level, which is an analogy of t', and use the selectivity of the *scarce q-gram* to estimate the selectivity of the query text t in the node.

Definition 3.15 (Scarce q-gram in a Node)**.** *A q-gram tk in node n is scarce if the proportion of objects containing tk in node n is less than a threshold ϵ.*

A scarce q-gram is actually a short identifying substring, because: (1) the selectivity of the scarce q-gram in a node is close to the selectivity of the original query in the same node because its value is small and is an upper bound of the true selectivity, and (2) q-grams are short.

For each q-gram of string t, the number of objects containing the q-gram in node n can be retrieved from the node-level inverted index. Among all the retrieved values in node n, we use $C_n^{\perp}(t)$ to denote the minimum value for the q-grams of t. Similarly, we use $C_n^{\perp'}(t)$ to denote the minimum value for the positional q-grams of t in node n.

As stated in Equation 3.1, we use $S_s(n,t)$ to denote the *substring selectivity* of string t in node n. If $(C_n^{\perp}(t)/|Obj(n)|) < \epsilon$, there exist scarce q-grams in node n for t. Then we set the value of $S_s(n,t)$ to $C_n^{\perp}(t)$. Otherwise, we set the value of $S_s(n,t)$ to $ME_s(n,t)$. Therefore, we get:

$$S_s(n,t) = \begin{cases} C_n^{\perp}(t) & \text{if } (C_n^{\perp}(t)/|Obj(n)|) < \epsilon \\ ME_s(n,t) & \text{otherwise} \end{cases}$$

By integrating the scarce q-gram with the Markov Estimator, the *prefix selectivity*

of a string t in node n is:

$$S_p(n,t) = \begin{cases} \min(C_n^{\perp}(t), C_n^{\perp'}(t)) & \text{if } ((C_n^{\perp'}(t)/|Obj(n)|<\epsilon) \\ & \&(C_n^{\perp}(t)/|Obj(n)|<\epsilon)) \\ C_n^{\perp'}(t) & \text{else if } C_n^{\perp'}(t)/|Obj(n)|<\epsilon \\ C_n^{\perp}(t) * \Re & \text{else if } C_n^{\perp}(t)/|Obj(n)|<\epsilon \\ ME_p(n,t) & \text{otherwise} \end{cases}$$

The selectivity is estimated for SPR and SS queries only, but not for SAP and SAS queries because additional information is required [85], which will bring more storage cost.

3.5.2 Reuse of Query Results

Besides the selectivity estimation, our incremental processing provides another optimization opportunity: reusing the result of the relaxation in earlier phases to accelerate the processing of a relaxed query.

Rule I. Reusing final results of the relaxation in earlier phases. The *inter-relaxation relationships* (Observation 1) indicate the inclusion relationships between relaxation at different levels. Therefore, we can reuse the less relaxed query's results as a part of the more relaxed query.

Rule II. Reusing common intermediate results. Recall Section 3.4, all types of relaxation are processed over our one-size-fits-all index, many intermediate results among queries can be reused as well, including:

a. Reusing the snippets (Definition 3.11). When the spatial region is relaxed, the SPR query is applied. Suppose that the snippet of node n is $NS(n, Q_p) = (n.hc,\ Obj(n),\ Obj(n, Q_p))$ for a SP query Q_p, it becomes $NS(n,\ Q_p^r) = (n.hc,\ Obj(n),\ Obj(n, Q_p^r))$ for the relaxed SPR query Q_p^r. When processing node n in Q_p^r, we only need to verify the objects in $(Obj(n, Q_p^r) \setminus Obj(n, Q_p))$.

When the prefix constraint is relaxed, the SS, SAP and SAS queries are applied. Since they share the same spatial region, the intersecting nodes remain the same.

In addition, the snippet of each intersecting node remains unchanged. Thus, this information can be reused.

b. Reusing filtering results. As shown in Table 3.4, we notice that (1) SP and SPR queries share the same PoF and CF filters; (2) SP and SS queries share the CF filter when the query length is greater than $UB_{p^\top,0,q_p}$; (3) SAP and SAS queries share the CF filter when the query length is greater than $UB_{p^\top,0,q_p}$. These filtering results can be reused.

c. Reusing the result of an SP query in the selectivity estimation. We can use the number of results of an SP query as the lower bound when estimating the selectivity of the SPR and SS queries (if needed), instead of computing the selectivity from scratch for these types of relaxation.

d. Reusing SPR's selectivity to estimate SS's selectivity. Based on Equation 3.3, $ME_s(n,t)$ can be computed using $ME_s(n,t) = ME_p(n,t)/\Re$. Since $ME_p(n,t)$ is already computed in estimating SPR's selectivity, we can reuse $ME_p(n,t)$ to compute the selectivity of the later SS query.

Example 3.10. *Suppose we have answered an SP query Q_p= (r, "star") in Example 3.6, we need to answer its relaxed SPR query Q_p^r=(r', "star"). By Observation 3.1, we obtain $Res(Q_p) \subseteq Res(Q_p^r)$ and $Res(Q_p) = \{O_5, O_6\}$. Thus, $Res(Q_p^r)$ must contain $\{O_5, O_6\}$. This shows the application of Rule I.*

According to the framework shown in Figure 3.5, before the processing of $Q_p.r$ is triggered, we estimate $S(Q_p^r)$ first. By Rule II.c, we only need to estimate the selectivity for the nodes that are not contained in $Q_p.r$. We are not required to estimate the selectivity for the nodes that are contained in $Q_p.r$, such as node 2.0.3.0, because we have obtained their results.

By Rules I and II.a, we do not revisit the objects in $Q_p.r$. We only need to visit object O_2, node 0.1 and 2.0.1. By Rule II.b, SP and SPR queries share the same

PoF filter. We notice that node 0.2 is removed by PoF_n previously. Consequently, we do not visit node 0.2 when we process Q_p^r.

3.6 Inter-query Optimization

Recall the framework in Section 3.2.3, we point out the relationship between a query and its appending query, and highlight the relationships between their results in Observation 3.2. Next, we present how to exploit the inter-query relationship to accelerate the processing of appending queries.

3.6.1 Processing Appending Queries

Before discussing how to process appending queries, we need to prove Observation 3.2, because it is a foundation of our methods. For a string query and its appending query, we observe the following properties.

Observation 3.3. *Given strings s,t,u, where s is a prefix of t, and an edit distance threshold τ, these properties hold:*

(1) If s is not a prefix of u, t is not a prefix of u.

(2) If s is not a substring of u, t is not a substring of u.

(3) If s is not an approximate prefix of u under τ, t is not an approximate prefix of u under τ.

(4) If s is not an approximate substring of u under τ, t is not an approximate substring of u under τ.

Proof. Properties (1), (2) and (3) can be easily derived from property (4). Here we prove property (4) by contrapositive. We use D(*,*) to denote the edit distance between two strings, which is calculated by:

$$D(a[0,i],b[0,j]) = min \begin{cases} D(a[0,i-1],b[0,j-1]) + d(a_i,b_j) \\ D(a[0,i-1],b[0,j]) + 1 \\ D(a[0,i],b[0,j-1]) + 1 \end{cases},$$

where $d(a_i, b_j)$=0 if a_i=b_j, and $d(a_i, b_j)$=1 if $a_i \neq b_j$. When we match a_i to b_j, one substitution is required if $a_i \neq b_j$. The substitution is not needed if we do not match a_i to b_j. Therefore, we get D(a[0,i],b[0,j])$\geqslant$D(a[0,i-1],b[0,j-1]).

By the definition of approximate substring, we suppose D(t,$u[i, j]$)=τ. We get D(t[0, |t|-1], u[i, j]) $\geqslant$ D(t[0, |t|-2], u[i, j-1]) $\geqslant$ ... $\geqslant$ D(t[0, |s|-1],u[i, j-(|t|-|s|)]) = D(s, u[i, j-(|t|-|s|)]).

As a result, we get a substring of u, whose edit distance to s is within τ. In other words, s is an approximate substring of u under the edit distance threshold τ. □

By Definition 3.8, the difference between a spatial query Q_ω and its appending query Q'_ω is: $Q_\omega.str$ is a prefix of $Q'_\omega.str$. Thus, we can easily derive the *inter-query relationship* (Observation 3.2) from Observation 3.3.

Based on the *inter-query relationship*, we process a query in the following way, as shown in Figure 3.5. Whenever a query Q'_ω is coming, we first check whether it is an appending query of the previous query Q_ω. If Q'_ω is an appending query of Q_ω, we process Q'_ω from where Q_ω stops. In addition, we use the results of Q_ω to form the candidate results of Q'_ω. Otherwise, Q'_p is a non-appending query. We process it as a fresh SP query.

Such *inter-query relationship* plus our incremental paradigm bring two immediate benefits: (1) When processing Q'_ω, instead of starting from scratch, we could start from the type of relaxation where Q_ω stops. Therefore, processing relaxation at previous levels can be skipped. (2) The result of Q_ω, i.e. $Res(Q_\omega)$, can be output to Q'_ω as candidates in the corresponding relaxation, and we only need to check whether each object in $Res(Q_\omega)$ satisfies the string condition for Q'_ω, as $Q_\omega.rng$=$Q'_\omega.rng$.

3.6.2 Merging Inverted Lists for Appending Queries

When the above reuse does not bring enough number of results for Q'_ω, we need to further relax Q'_ω and process it by adopting Algorithm 1. The most time consuming part is to get candidates by applying the filters shown in Section 3.4.1, i.e. to get the objects that appear at least m times in the given inverted lists. It is formalized as the *list merging problem* in [122, 87]. Note that the value of m corresponds to the minimum match threshold, which is shown in Table 3.4.

Now, our problem becomes: given a set of lists L_{old}, we have found the candidate set C_{old}, of which an id appears at least m times in L_{old}. When a new set of inverted lists L_{new} comes for the appending query (brought by the new q-grams), we need to find a set of candidates C_{new}, of which an id appears at least m' times in the set $(L_{old} \bigcup L_{new})$.

We propose a novel algorithm to address this problem, where our idea is to *reuse* the results computed for Q_w to efficiently merge the lists when processing its appending query Q'_w. When processing Q_w, we find these properties:

Property 3.1. *For each id in C_{old}, we have its number of occurrences in L_{old}, which is at least m.*

Property 3.2. *For all the ids not in C_{old}, they appear at most (m-1) times in L_{old}.*

Based on the above two properties, we process the ids in and outside C_{old} separately. There are three cases.

Case (1): $m' \geqslant (m+|L_{new}|)$. For each id in C_{old}, by Property 3.1, we do not need to visit L_{old} again and can directly check whether it has enough number of appearances in L_{new}. For all the ids not in C_{old}, by Property 3.2, they appear at most $(m\text{-}1+|L_{new}|)$ in $(L_{old} \bigcup L_{new})$, which is less than m'. Thus, they cannot be in C_{new}.

Case (2): $m \leqslant m' < (m + |L_{new}|)$. For each id in C_{old}, the same method is applied as Case (1). For the ids not in C_{old}, we present how to process them in Section 3.6.2.1.

Case (3): $m' < m$. We need to restart merging the inverted lists in $(L_{old} \bigcup L_{new})$ to get C_{new}.

3.6.2.1 Processing the ids not in C_{old} of Case (2)

We continue to show how to process the ids not in C_{old} of Case (2). Based on Property 3.2, we find:

Property 3.3. *For the ids not in C_{old}, if a minimum of m' occurrences is required in $(L_{old} \bigcup L_{new})$, they must appear at least $(m'\text{-}m+1)$ times in L_{new}.*

The MergeSkip algorithm [87] is proposed to efficiently merge lists using a heap structure. The heap is used to record the frontiers of the inverted lists. If a minimum of m occurrences is required, the heap returns the $(m)^{th}$ smallest record in each round, whose value is t. The records smaller than t are skipped in the inverted index because they cannot appear more than m times.

Similar to the MergeSkip algorithm, we maintain a heap H_{all} for frontiers of the lists in $(L_{old} \bigcup L_{new})$. In addition, we maintain another heap H_{new} for frontiers of the lists in L_{new}. In each round, H_{all} returns the $(m')^{th}$ smallest record, whose value is t_{all}, while H_{new} returns the $(m'\text{-}m+1)^{th}$ smallest record whose value is t_{new}. By Property 3.3, the records smaller than $\max(t_{all}, t_{new})$ are skipped because they cannot appear more than m' times.

Example 3.11. *Figure 3.6 shows an example of merging lists for the ids not in C_{old}. Given $m = 3$, we need to get the candidates with minimum $m' = 4$ occurrences in $(L_{old} \bigcup L_{new})$. We use H_{all} to store the frontiers of all the lists in Figure 3.6(b) and H_{new} to store the frontiers of the lists in L_{new} in Figure 3.6(c). In round one, H_{all}*

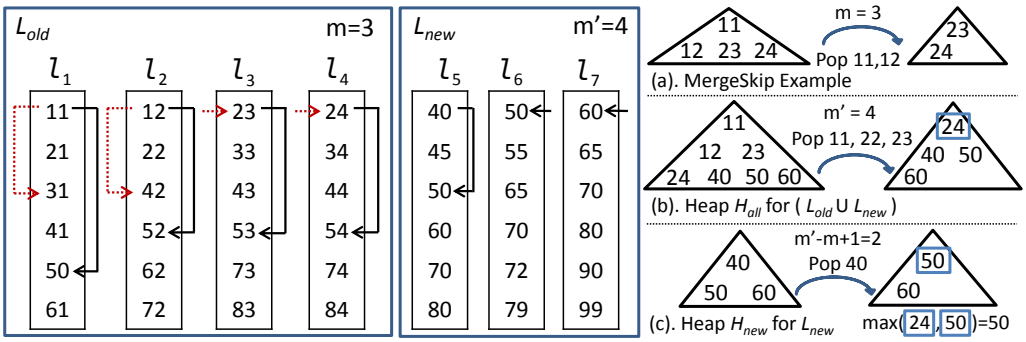


Figure 3.6: Merging Lists of Case (2)

outputs the 4^{th} *smallest number, which is 24.* H_{new} *outputs the* 2^{nd} *smallest number, which is 50. Since* $\max(50,24) = 50$*, the records smaller than 50 are skipped, as the black arrows on the right show. In contrast, only the records smaller than 24 are skipped in MergeSkip algorithm.*

When the new inverted lists are added for the appending query, the MergeSkip algorithm merges the lists from scratch, while our algorithm optimizes this process by reusing the intermediate results computed from the previously-formed query. Therefore, we do not need to merge the lists from scratch for Cases (1) and (2).

3.7 Experiment

In this section, we aim to study four issues: (1) how the intra-query optimization boosts the performance of `INSPIRE` for *non-appending* queries, i.e. query reuse and selectivity estimation; (2) how the inter-query optimization accelerates the processing of the *appending queries* in `INSPIRE`; (3) compare `INSPIRE` against existing work for each specified type of relaxation to show that there is a need for a single, unifying framework to support spatial prefix query and its different types of relaxation; (4) compare `INSPIRE` against existing map services to show the effectiveness of our approach in the context of local search.

Table 3.5: Statistics of the used data sets in Chapter 3

Property	*OSM*	*GNIS*	*SGP*
Number of records	1,616,295	2,078,921	12,918,933
Size (MB)	103.7	136.7	867.2
Max text length (characters)	173	104	200
Average text length (characters)	18	19	19

3.7.1 Experiment Setting

Data sets: We report results over three real POI data sets in the United States: *OSM*, *GNIS* and *SGP*. *OSM* is downloaded from the OpenStreetMap project[1] which contains around 1.6 million POIs. *GNIS* is extracted from the Geographic Names Information System[2] and contains the geographic name usage in the U.S. Government, including about two million records. *SGP* is obtained from the SimpleGeo's Places[3] and has thirteen million records. *SGP* is then merged and maintained by Factual, which is a popular location service platform. The statistics are shown in Table 3.5.

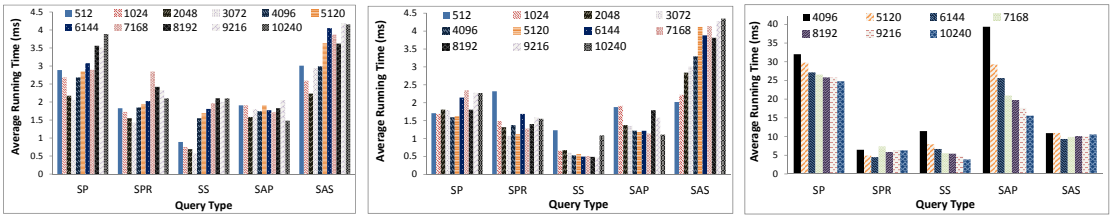
Query set: To ensure that the queries have nonempty results, we generate queries in the following way: we randomly select 1000 objects with the length of the textual content larger than 5. Unless otherwise specified, we use the first keyword as the query text to make it compatible with some existing methods that can only process queries at the word level. The result size threshold θ is set to 10. The edit distance threshold τ is set to 20% of the query length for SAP and SAS queries.

We normalize the latitude and longitude into [0, 1]. Thus, the query range is denoted by the percentage of coverage in the geographic space. Six spatial ranges with the selected object as the center are generated: 0.005^2, 0.01^2, 0.02^2, 0.04^2, 0.06^2 and 0.08^2. The first (last) range is at a town (state/province) level. By default, we

[1] http://www.openstreetmap.org/

[2] http://geonames.usgs.gov/domestic/download_data.htm

[3] http://s3.amazonaws.com/simplegeo-public/places_dump_20110628.zip



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.7: Impact of the max capacity

use the 0.01×0.01 range at a city level and expand the query region by doubling the area in the SPR query.

Comparisons: `TAS` [13] and `FEH` [74] are proposed to provide type-ahead search in spatial databases. Besides, `LBAK` [4] and `MHR` [140] are proposed to allow approximate keyword matching in spatial databases. We compare with `TAS` for SP queries, `FEH` for SP and SS queries, `LBAK` and `MHR` for SAP and SAS queries. In particular, `TAS` and `LBAK` are memory resident while `FEH` and `MHR` are disk resident. The parameters of the baseline algorithms are set to the default values in the original papers. Our methods are implemented in JAVA. All experiments are run on a Quad-Core AMD Opteron Processor 8356 @ 2.29 GHz with 128 GB main memory. Queries are run under a heap with the max size of 4 GB.

Parameter Choice:

According to [79], the performance of a Quadtree depends on the tiling level, which in turn depends on the max capacity of a Quadtree node. So we study the performance of each type of relaxation w.r.t. different choices of the max capacity. We test values from 512 to 10240 for *GNIS* and from 512 to 10240 for *SGP*, as shown in Figure 3.7. The time for the values from 512 to 3072 is omitted on *SGP* as it is much longer than that of the rest.

We take all types of queries into consideration when choosing an appropriate max capacity. For each type of query, the max capacity with the shortest running time

Table 3.6: Index Construction Time and Index Size

Index	*OSM*		*GNIS*		*SGP*	
	Time(mins)	Size(MB)	Time	Size	Time	Size
`INSPIRE`	2.7	346	3.6	448	21.5	3857
`TAS` [13]	2.0	150	3.4	194	20.1	1445
`FEH` [74]	5.1	306	7.3	405	33.6	2704
`MHR` [140]	4.2	663	7.4	857	37	5332
`LBAK` [4]	1.0	253	1.4	291	9.5	2541

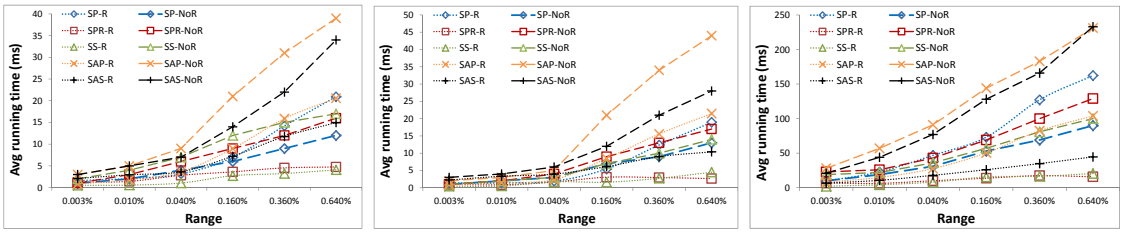
is assigned a score of 1 while the scores for the rest are computed as the proportion of the running time to the shortest one. The score of the overall performance for each max capacity is the summation of its scores for all the five types of queries. We choose the parameter with the lowest score, which indicates the best overall performance. As a result, we set the max capacity to 3072, 3072 and 10240 for *OSM*, *GNIS* and *SGP* respectively.

In addition, the length of q_c-gram and positional q_c-gram is set to 3 and 2 respectively, which is the same as the setting in [77]. The reserved position $p^\top$ is set to 9. By Definitions 3.12 and 3.13, up to four edit operations can be supported by PrF. If there is an error in every five characters, PrF can be applied when the query length is less than 20.

Index Construction: We build our index based on the above parameters. Table 3.6 shows the construction time and the size of our proposed index for the chosen parameters and the compared indexes. Overall, `LBAK` builds the index in the shortest time, while `TAS` has the smallest index size. Our index is around the medium in the term of construction time and the index size.

3.7.2 Comparison between Variants of `INSPIRE`

In Sections 3.5 and 3.6, the intra-query optimization and inter-query optimization are proposed to accelerate the processing of non-appending and appending queries respectively. So we would like to study the effectiveness of these optimizations.



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.8: Relaxation of the SP query

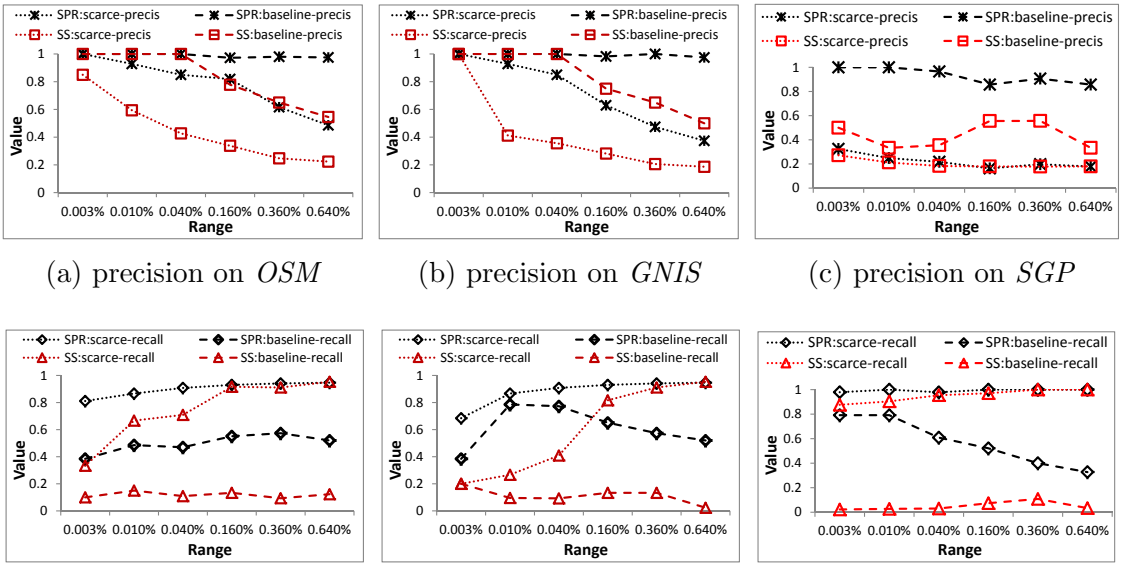
3.7.2.1 Intra-query Optimization

We first study the impact of each of the two intra-query optimization techniques to accelerate the processing of the *non-appending queries*: result reuse and selectivity estimation (in Section 3.5). Note that, for each technique under test, the other one is turned on by default.

Result Reuse: We compare the running time of the relaxed queries with reuse to that without reuse. As shown in Figure 3.8, we find that the performance of each relaxed type boosts up by four times for *SGP* on average. The SP query takes more time for the reuse version as common intermediate results and processing are saved. Compared to the extra cost of the SP query, it is worth applying this optimization as the benefit that it brings is significant. For *OSM* and *GNIS*, the performance boosts up by three times on average.

Selectivity Estimation: We compare our method using the scarce q-grams to the baseline method, which uses the Markov Estimator as described in Section 3.5.1.1.

Note that a query is triggered if its estimated selectivity is greater than the result size threshold θ. Therefore, the terms positive and negative are defined as whether the estimated selectivity is greater than or less than θ respectively. Accordingly, when both the true selectivity and the estimated selectivity are greater than θ, we



(a) precision on *OSM* (b) precision on *GNIS* (c) precision on *SGP*

(d) recall on *OSM* (e) recall on *GNIS* (f) recall on *SGP*

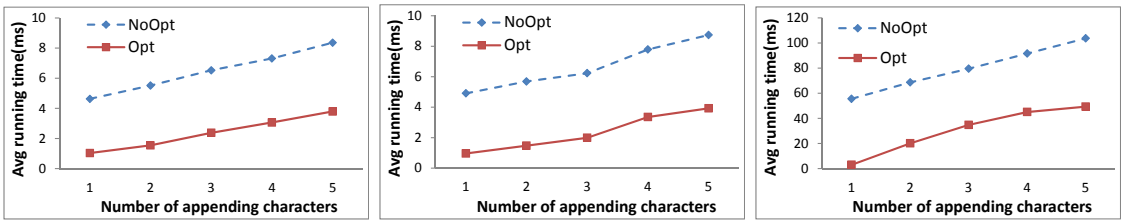
Figure 3.9: Precision and Recall

refer to it as *true positive*. The scarce threshold ϵ is set to 0.01.

Figure 3.9 shows the precision and recall for the SPR and SS queries in different query ranges. Overall, our approach has a much higher recall than the baseline method, and it is at least 0.8 across all query ranges for *SGP*. In contrast, the recall of the baseline is low for SS queries so that many queries that should be processed are skipped. The baseline approach has higher precision than ours, because it underestimates the selectivity when a query contains scarce q-grams. However, a potential problem of such high precision is to bring more time complexity in processing the relaxed query at the next level as they may lose the possibility to reuse the results got in earlier phases.

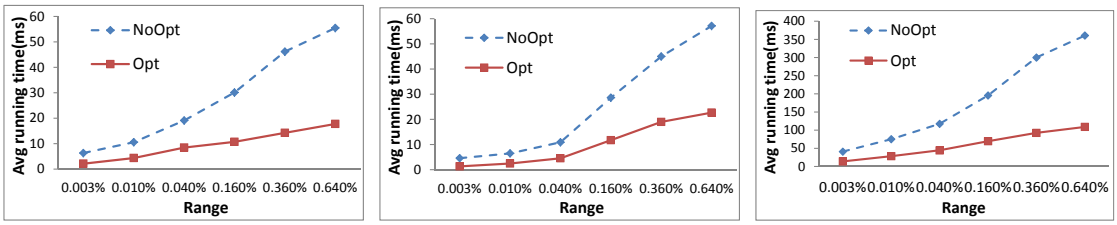
3.7.2.2 Inter-query Optimization

In this part, we would like to study the impact of the inter-query optimization (proposed in Section 3.6) to accelerate the processing of the *appending queries*. The non-optimized approach is the one treating an appending query as a fresh query.



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.10: Impact of inter-query optimization w.r.t. appending characters

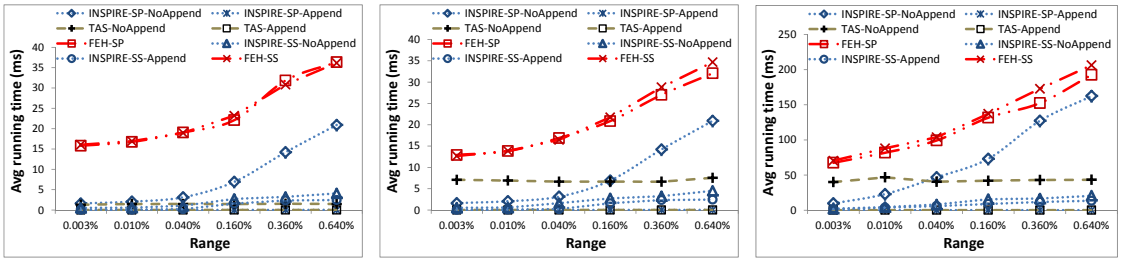


(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.11: Impact of inter-query optimization w.r.t. query ranges

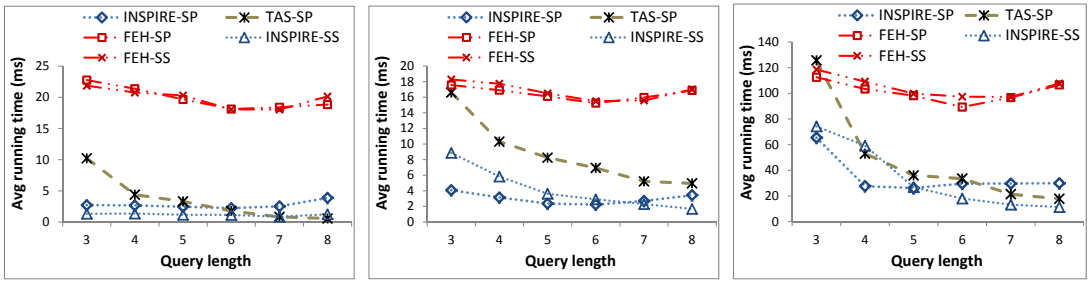
We first investigate the impact of the number of appending characters, ranging from 1 to 5. From Figure 3.10, We find: (1) The processing time increases as the number of appending characters increases, because more inverted lists need to be merged to get candidates. (2) On average our inter-query optimization brings around three times acceleration on the three data sets.

Next, we study how the performance of the appending query varies with respect to different query ranges, while the number of appending characters is randomly selected from one to five for each query to simulate real cases. From Figure 3.11, we find that the method with optimization scales better than the one without optimization on *SGP*. This same finding exists on *OSM* and *GNIS* as well.



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.12: Comparisons of SP & SS Queries w.r.t. query ranges



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.13: Comparisons of SP & SS Queries w.r.t. query ranges

3.7.3 Comparison to Other Approaches

In this subsection, we compare `INSPIRE` (which is equipped with all the aforementioned optimizations) with existing mothods. However, all state-of-the-art work only studies a certain type of relaxation for the spatial prefix query (see Chapter 2.2 for more details). As such, we have to compare `INSPIRE` with each individual type of relaxation even though it is not a fair comparison for a general framework. For each type of relaxation, we divide the query set into two parts: appending queries and non-appending queries.

3.7.3.1 SP & SS Queries

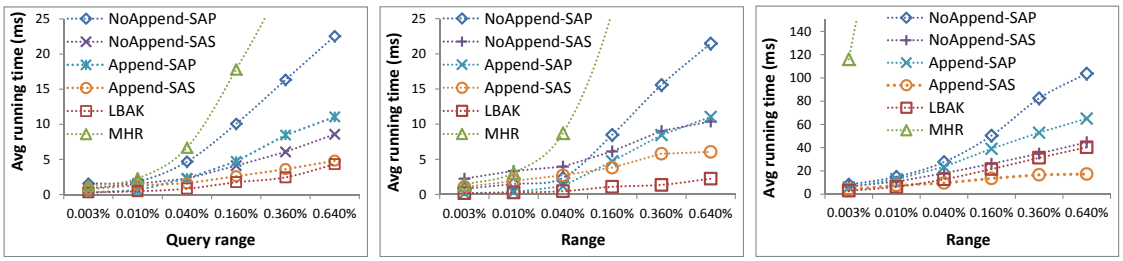
We compare with `TAS` [13] for SP queries, and `FEH` [74] for SP and SS queries. Figure 3.12 plots the average running time in different query ranges. We find: (1) `INSPIRE`

outperforms TAS when the query range is small, while TAS has better performance when the query range increases. This is because TAS retrieves candidates on the textual dimension first and then verifies candidates on the spatial dimension. However, INSPIRE performs in the opposite way. TAS achieves the best performance on SP queries in some cases as it is optimized to answer the spatial prefix query. It uses a trie-like index so that it works well but can only answer SP queries. Although INSPIRE loses to TAS on SP queries in those some, it can handle all types of queries. (2) Besides, INSPIRE outperforms FEH on both SP and SS queries. (3) Moreover, the running time of appending queries is shorter than that of non-appending queries. In particular, for SP queries, the running time of appending queries is almost 0 because the results of previously-formed queries can be used directly.

Furthermore, we investigate the impact of query lengths. Figure 3.13 demonstrates the query performance for lengths ranging from 3 to 8. INSPIRE outperforms TAS when the query text is short. This is because TAS uses a trie-like index to retrieve candidate objects that satisfy the prefix condition. Then all the candidate objects are verified against the spatial condition one by one to get the final results. When the query text is short, the candidate size could be huge. Thus the verification time is long. We also find that INSPIRE outperforms FEH, mainly because FEH needs to retrieve inverted entries at each level of its spatial index.

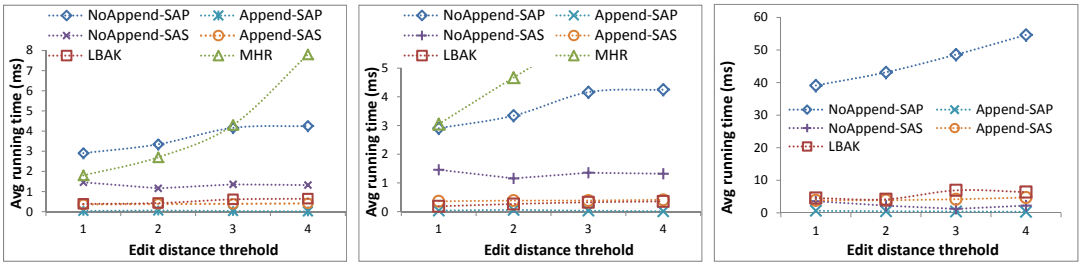
3.7.3.2 SAP & SAS Queries

For SAP and SAS queries, no existing work has addressed the search-as-you-type problem at the q-gram level, so no direct comparison can be made. Instead, we compare with two approaches (LBAK [4] and MHR [140]) at the word level. They require users to type at least one full word for a query. Accordingly, we form queries by extracting the first word from each query of our query set. Again, we note that it is not a fair comparison because INSPIRE works at the q-gram level.



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.14: Comparisons of SAP & SAS Queries w.r.t. query ranges



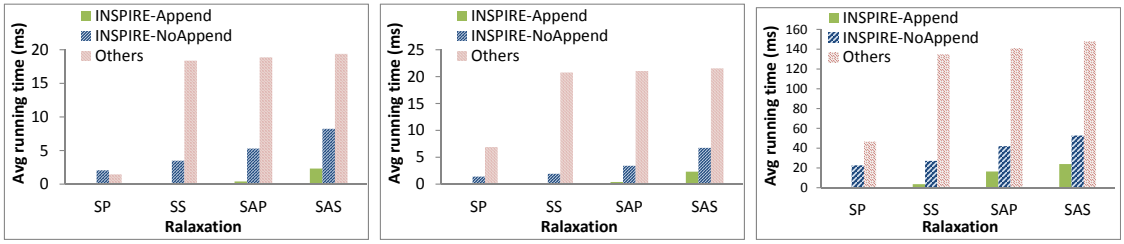
(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.15: Comparisons of SAP & SAS Queries w.r.t. edit distances

Figure 3.14 shows the average running time of different query ranges. `LBAK` achieves the best performance while `INSPIRE` gives intermediate performance. One reason is that `INSPIRE` is proposed in a more general setting, which handles approximate queries in the search-as-you-type paradigm[4]. Another reason is that `LBAK` is memory-based, thus disk access is not required. Although it is not a fair comparison, the performance of `INSPIRE` is at the same level as that of the state of the art.

Impact of edit distance threshold: Next, we test the performance for different values of the edit distance threshold τ. We randomly select objects with the length of the first keyword greater than 17 from both data sets. Then we add one, two, three, and four errors respectively to the query to evaluate the performance. The errors are randomly generated by skipping letter, doubling letters, reversing

[4] `LBAK`, for example, cannot handle both exact and approximate substring queries.



(a) *OSM* (b) *GNIS* (c) *SGP*

Figure 3.16: Efficiency

letters, replacing letter, or inserting letter.

Figure 3.15 shows the average running time for different τ values. For the non-appending queries, `LBAK` achieves the best result. For the appending queries, `INSPIRE` is the best because the results of the previously-formed queries can be reused. We do not plot the results of `MHR` because the minimum running time for `MHR` is 280 milliseconds.

3.7.3.3 Overall Performance

At the framework level, we would like to see `INSPIRE` 's overall performance in a real usage scenario, where we do not know which type of relaxation is sufficient to obtain enough number of results. Since all of the existing approaches study a single type of relaxation, we construct a baseline framework as an integration of the state of the art for each type of relaxation, and compare with `INSPIRE` equipped with the proposed inter-query and intra-query optimizations. In particular, `TAS` is tailored for SP queries, while `FEH` is tailored for SS queries and `LBAK` is tailored for SAP and SAS queries.

Figure 3.16 shows the accumulated time when different types of relaxation stop. `INSPIRE` outperforms such an integrated framework. Especially for appending queries, `INSPIRE` handles them almost for free due to the various result reuse algorithms that we have proposed. In contrast, a simple integration of existing methods does not

have such benefit, because the existing methods answer one type of particular query using an optimized index structure. Although our method loses for some particular types of queries, our INSPIRE framework outperforms the existing methods as a system to provide relaxation for spatial prefix query. This finding confirms the need for a single, unifying framework to support the spatial prefix query and its different types of relaxation.

3.7.4 Effectiveness Study

We conduct an effectiveness study on local search to simulate the real-life search scenario. Queries are issued in a small spatial region such as a city. We test five online map services, including Google Maps, Bing, Yahoo!, Here and OpenStreetMap. Only Google and Yahoo! provide the instant search feature. We use the *SGP* data set in `INSPIRE` for this effectiveness study.

For the query set, we first find tourist attractions from the five cities with the largest population in the USA: New York (NY), Los Angeles (LA), Chicago (CH), Houston (HO) and Philadelphia (PH). Then we extract the texts from these tourist attractions and generate four types of queries:

Type I. the query is a prefix of the text;

Type II. the query is a substring of the text;

Type III. the query is a prefix of the text, with typos;

Type IV. the query is a substring of the text, with typos.

For each city, four types of queries are formed on the attractions in the city, and each type contains ten queries.

Evaluation Method: We invite twenty participants to take part in the task of scoring the top-5 results from the three systems. To conduct a fair evaluation, the source of each result is kept anonymous. The participants are asked to score the quality of each result by using the Cumulated Gain-based evaluation [71] metric (from

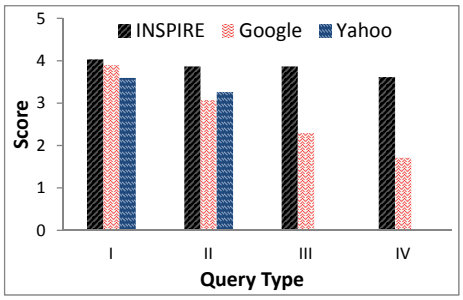


Figure 3.17: Effectiveness

0 to 5 points, 5 means the best while 0 means the worst). Each participant grades the results of three cities. Therefore, for each result, there are twelve participants grading it. The average score is presented in Figure 3.17.

Results: We first report the summarized result, which is shown in Figure 3.17. For query Types I and II, the three systems perform equally well. For Types III and IV, `INSPIRE` outperforms Google Maps. The results of Yahoo! Maps are not shown because it does not support error-tolerant search. The reason why `INSPIRE` performs better is that `INSPIRE` in essence is able to answer error-tolerant instant search. Google Maps, though, probably adopts a global search strategy to serve the users from all over the world.

We then explain the results using some selected examples, which are listed in Table 3.7. For Types I and II, the three maps return correct results in most cases. Still, some results are not related to the issued query. We show two sample results for Q_3. From the column on the right side, `INSPIRE` returns a POI in Houston. The query text is an approximate prefix of the text of the POI. However, the POIs returned from the other two systems are spatially far away from the query region. For Types III and IV with typos, `INSPIRE` can correct the typos in the queries and return reasonable answers. In contrast, Google Maps does not always perform well. For Q_5 and Q_7, Google Maps again returns distant POIs from the query region, probably because their names match the queried text. This approach is a double-edged sword.

Table 3.7: Query Examples

Type I	Q_1:(CH, "navy pier")	Q_2:(NY, "times square")
Inspire	navy pier in CH	times square in NY
Google		
Yahoo!		
Type II	Q_3:(HO, "race park")	
Inspire	sam houston race park in HO	ace park & ride in HO
Google		delaware park in Wilmington
Yahoo!		seven flags race park in Dickson
Type III	Q_4:(LA, "doger stadium")	Q_5:(NY, "yangkee")
Inspire	dodger stadium in LA	yankee stadium in NY
Google		yangkee logistics in Singapore
Type IV	Q_6:(HO, "otdoor theatre")	Q_7:(LA, "librart")
Inspire	miller outdoor theatre in HO	library tower in LA
Google		librarti sa in Switzerland

On one hand, it suggests distant objects that possibly match the query terms, which is pretty good if the user is searching on a global scale. On the other hand, it dwarfs the approximate answers which are local. Depending on users' intention, this may or may not be helpful. However, in our context of local search, the local answers surely play a more important role.

From the above effectiveness study, we find: (1) the search strategy proposed in `INSPIRE` fulfills the need for context-aware local search; (2) `INSPIRE` is a good complement to online Web services on context-aware local search. When a user conducts a local search, the user would continuously zoom in a particular spatial region in the viewport. Therefore, the local search pattern can be easily detected and the local search can be optimized using `INSPIRE` to improve the user experience.

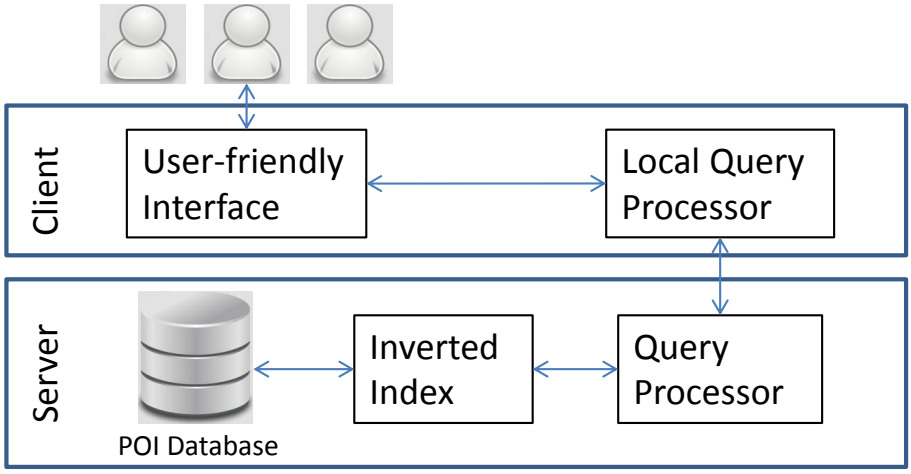


Figure 3.18: System Architecture

3.8 Implementation of INSPIRE

In this section, we present the implementation details of INSPIRE. We adopt the client-server architecture to build the system, which is shown in Figure 3.18. On the client side, when a user performs a query, he/she would first zooms in the particular spatial region that he/she wants to search. Together with the text typed by the user, the query is sent to the server for processing. After the query is processed, the results are sent back and displayed. An online system [147] can be found here[5].

On the server side, INSPIRE contains two major components: (1) the one-size-fits-all index in Section 3.3 that can answer the spatial prefix query and support the computation of its different types of relaxation; (2) a query processor to handle queries issued from the client side. It adopts the incremental search strategy to process a spatial keyword query, which is presented in Section 3.2.3. The server is built using JSP and Apache Tomcat.

On the client side, INSPIRE includes two main components: (1) a user-friendly map interface to provide interactive search and result display, which is implemented using Google Maps API; (2) a local query processor to answer appending queries

[5] http://dbgpucluster-2.d2.comp.nus.edu.sg:8080/MESA/

if the appending queries can be answered using the local cached results to save computational cost. The client side is implemented using JavaScript.

3.8.1 User-friendly Interface

The user-friendly Interface is used to provide interactive search and display results. For the interaction part, we provide a control panel to let users specify the input parameters if they wish. Therefore, customized search can be achieved. Advanced users can issue their queries on demand, while common users can just use the default setting to issue queries. For the result display part, we use different colors to visualize the results returned from different types of relaxation. As a result, results can be differentiated. This is motivated by the fact that users usually prefer the exact matching results to the approximate matching results. Besides, users can click the spatial objects shown on the map for further information.

3.8.2 Local Query Processor

As described previously, the autocompletion paradigm allows additional characters to be appended after the initial query. To save the communication cost between the client and the server, we cache the results of the previous-issued query. When an appending query comes, before the client posts the query to the server, the local query processor first checks whether the local cached results can answer the appending query. If yes, the answer set can be reused and the appending query is not necessary to be sent to the server; otherwise, the client needs to send the appending query to the server. Then the server side processes the appending query and returns the new results to the client.

All in all, we group these key components into a system to provide the spatial prefix search and its different types of relaxation. The techniques, like spatial q-gram, two-level inverted index and the unifying search strategy, are proposed to

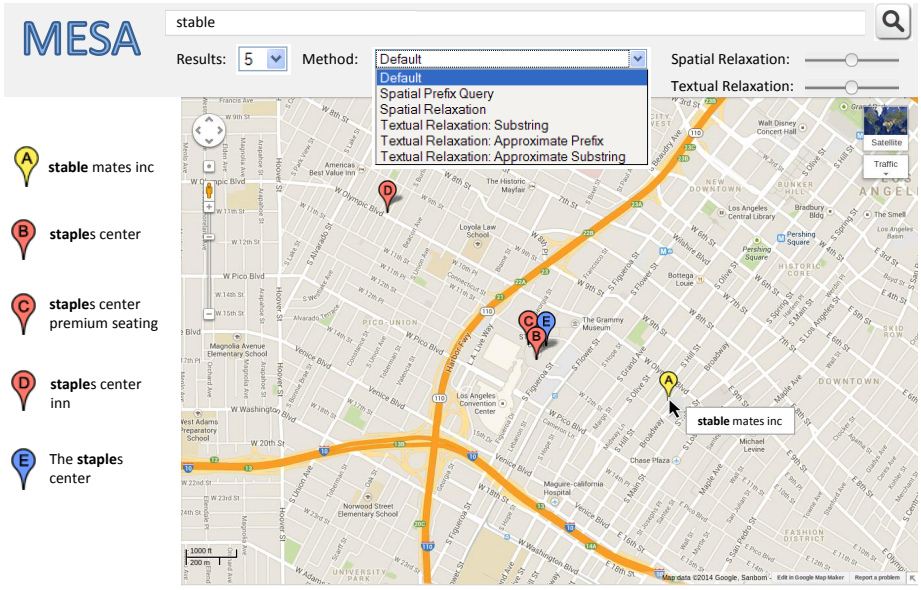


Figure 3.19: Interface of INSPIRE

make the spatial prefix query and its various types of relaxation more efficient and effective. Each of those techniques is an indispensable component of our system, whose ultimate goal is to efficiently and effectively process fuzzy type-ahead query in the context of spatial keyword search.

3.8.3 System interface

Then we would like to demonstrate how INSPIRE handles spatial prefix search and its different types of relaxation effectively and efficiently, and how it enhances user experience by providing the error-tolerant autocompletion feature in the context of spatial keyword search. Users will be able to experience how INSPIRE facilitates spatial keyword search because they do not need to type a full query text to issue a query. We use the datasets described in Section 3.7 in the demonstration.

The interface is shown in Figures 3.19. It consists of two parts: a control panel (upper part) and a display panel (lower part). The control panel is composed of four components: (1) a query text field to let users type queries; (2) a drop-down list to

input the number of required results; (3) anther drop-down list to specify the type of relaxation; and (4) two slider bars to indicate the degrees of relaxation. If the type of relaxation is specified, INSPIRE directly performs the particular type of relaxation. The slider bars are used to indicate what degrees of relaxation a user prefers. For example, users can decide how much to expand in the spatial region and what edit distance threshold to use in approximate prefix/substring queries.

Besides, the display panel is composed of two parts: a result list and a map view. In the result list, the results of a query are shown in different colors. The color indicates that the result is returned from which type of relaxation. In addition, the results returned from the relaxation in earlier phases are ranked higher than those results returned from the relaxation in later phases because users usually prefer the exact matching results to the approximate matching results. In the map view, the returned objects are also shown on the map. When users click a result on the map, its further information pops up.

To summarize, we showed the three features of INSPIRE.

1. INSPIRE can answer each type or degree of relaxation on its own. If a user specifies a particular type of relaxation for a query, INSPIRE can directly perform the type of relaxation. This feature can be shown by choosing different types of relaxation to be performed on the control panel.

2. INSPIRE provides efficient error-tolerant autocompletion. We show that the results can be returned in real time to fulfill real-time user-computer interaction requirement.

3. INSPIRE equips a user-friendly interface to interact with users and visualize the results in a proper way. Using this interface, customized search can be supported to meet the needs of different users. We present this feature by

showing results of queries issued in different places. For example, advanced users can control which type of relaxation to perform for a particular query.

3.9 Summary

In this chapter, we proposed INSPIRE, a general framework, which adopts a unifying strategy for processing different variants of spatial keyword queries. We observed that relaxation can be done by expanding the spatial region or loosening the prefix matching constraint. To maximize query reusability, we adopted an incremental way to perform relaxation. Then we built a one-size-fits-all index to support all types of relaxation. To accelerate query processing, we proposed our intra-query optimization covering common result reuse and selectivity estimation. Moreover, we observed that the search-as-you-type paradigm allows appending characters after the initial query, so we proposed the inter-query optimization to reuse the results of the initial query in processing its appending query, instead of processing it from scratch. As a result, INSPIRE is able to decide the most appropriate relaxation for a spatial prefix query, and outperforms most existing work.

Chapter 4

LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index

In the previous chapter, we study the interactive method of spatial keyword search. However, spatial keyword search is not applicable when text is not available. In such cases, new search methods are proposed to search complicated objects such as images, because conventional keyword search might not be used easily to describe such complicated objects. As a result, new search methods, e.g. "search by image" and "voice search", are proposes to support the retrieval of such complicated objects. In this chapter, we focus on the spatial image search, which is essentially modeled as the nearest neighbor search in high-dimensional spaces. We attempt to present an efficient solution to approximate nearest neighbor search in high-dimensional spaces.

Due to the "curse of dimensionality" problem, it is very expensive to process the nearest neighbor (NN) query in high-dimensional spaces; and hence, approximate approaches, such as Locality-Sensitive Hashing (LSH), are widely used for their theoretical guarantees and empirical performance. Current LSH-based approaches target at the ℓ_1 and ℓ_2 spaces, while as shown in previous work, the fractional distance met-

rics (ℓ_p metrics with $0 < p < 1$) can provide more insightful results than the usual ℓ_1 and ℓ_2 metrics for data mining and multimedia applications. However, none of the existing work can support multiple fractional distance metrics using one index. In this chapter, we propose LazyLSH that answers approximate nearest neighbor queries for multiple ℓ_p metrics with theoretical guarantees. Different from previous LSH approaches which need to build one dedicated index for every query space, LazyLSH uses a single base index to support the computations in multiple ℓ_p spaces, significantly reducing the maintenance overhead. Extensive experiments show that LazyLSH provides more accurate results for approximate kNN search under fractional distance metrics.

4.1 Motivation

State-of-the-art kNN processing techniques have been proposed for low-dimensional cases. However, due to the "curse of dimensionality", the same techniques cannot be directly applied to high-dimensional spaces. It was shown that conventional kNN processing approaches become even slower than the naive linear-scan approach [42]. One compromise solution is to adopt the approximate kNN technique which returns k points within distance cR from a query point, where c is the approximation ratio and R is the distance between the query point and its true (k)th nearest neighbor. The intuition is that in high-dimensional spaces, approximate results are good enough for most applications.

To process approximate kNN queries, several methods have been proposed [8, 68, 3, 98], among which, locality-sensitive hashing (LSH) [68] is widely used for its theoretical guarantees and empirical performance. In essence, the LSH scheme is based on a set of hash functions from the *locality-sensitive hash family* which guarantees that similar points are hashed into the same buckets with higher probabilities than

Table 4.1: Classification accuracy

Dataset	Classification accuracy (%)						
	Real 1NN	LazyLSH (Approximate 1NN)					
	$\ell_{1.0}$	$\ell_{0.5}$	$\ell_{0.6}$	$\ell_{0.7}$	$\ell_{0.8}$	$\ell_{0.9}$	$\ell_{1.0}$
Ionos	90.9	**92.0**	91.7	91.7	91.7	91.7	91.5
Musk	93.5	**94.0**	93.8	93.5	93.4	93.4	93.5
BCW	92.8	**93.3**	**93.3**	93.1	93.0	92.6	92.8
SVS	67.5	67.8	**68.9**	67.8	67.4	67.2	67.5
Segme	91.9	92.1	92.1	**92.4**	92.3	92.1	91.9
Giset	96.2	94.9	95.7	96.4	96.4	**96.8**	96.5
SLS	90.0	87.8	88.3	88.7	89.2	**90.0**	89.8
Sun	9.5	9.0	9.3	9.3	9.4	9.4	**9.5**
Mnist	96.3	95.1	95.4	95.7	95.9	96.0	**96.2**

dissimilar points. The LSH scheme was first proposed by Indyk et al. [68] for the use in the binary Hamming space, and later was extended for the use in the Euclidean space by Datar et al. [42] based on the p-stable distribution.

It was observed that the effectiveness of high-dimensional search is sensitive to the choice of distance functions [2]. Although the Manhattan (ℓ_1) and Euclidean (ℓ_2) metrics are widely used, it was shown that ℓ_p metrics with $0 < p < 1$, called *fractional distance metrics*, can provide more insightful results from both the theoretical and empirical perspectives for data mining applications [2, 39] and content-based image retrievals [67]. Furthermore, it was shown that the optimal ℓ_p metric is application-dependent and need to be tuned or adjusted for each application [2, 39, 67, 54].

As an example, Table 4.1 shows the accuracy of the kNN classifier [40] under different ℓ_p metrics. We test *Mnist* [84], *Sun* [47] and seven datasets from the UCI ML repository[1]. The ground-truth classification results are provided by the datasets themselves. For each query point, we retrieve its nearest neighbor and assign it to the same class tag as its nearest neighbor. For ℓ_p metrics ($0.5 \leqslant p \leqslant 1$), we compute

[1] http://archive.ics.uci.edu/ml/
The used datasets are: Ionosphere (Ionos), Musk, Breast Cancer Wisconsin (BCW), Statlog Vehicle Silhouettes (SVS), Segmentation (Segme), Gisette (Giset), Statlog Landsat Satellite (SLS).

the approximate 1NN using our LazyLSH technique. For comparison, we also show the results of the true 1NN in the ℓ_1 space. We highlight the highest accuracy for LazyLSH in bold font. The results indicate that the best classification result may be obtained using different fractional distance metrics for different datasets. There is no way to know which fractional distance is optimal for a specific dataset. Therefore, before implementing a system, we need an approach that can explore the data using different distance metrics, such that we can select the proper one to achieve the best mining results.

Unfortunately, due to the lack of closed form density functions for p-stable distributions when $p \neq 1$ or 2, it is non-trivial to generate p-stable random variables and build an optimal index structure for fractional distance metrics. Moreover, the conventional approach of building one index for each possible value of p will incur very high cost in terms of computational time and space requirement (with the number of possible values of p being potentially infinite). To address this problem, we propose LazyLSH as an efficient mechanism to process approximate kNN queries in different ℓ_p spaces using only one single index.

LazyLSH builds an LSH index in a predefined ℓ_{p_0} space, which is referred to as the base space. Using this materialized index, LazyLSH can answer approximate kNN queries in a user-specific query space. The word "Lazy" is borrowed from the lazy learning algorithms [145] in which generalization beyond the training data is delayed until a query is issued. LazyLSH means that we do not build an index for every query space. Instead, we reuse the index constructed in the base space to answer queries in the query space. Our analysis shows that if two points are close in an ℓ_{p_1} space, then they are likely to be close in another ℓ_{p_2} space. We also find that a locality-sensitive hash function built in the base space is still locality-sensitive in the query space when certain conditions hold. With this observation, LazyLSH adopts the strategy of having "**one index for many fractional distance metrics**". Figure

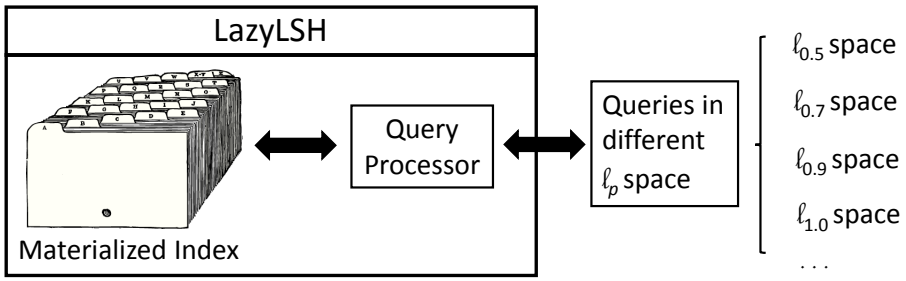


Figure 4.1: LazyLSH Overview

4.1 illustrates the basic idea. A single materialized LSH index is built using a specific distance function, based on which, we can approximately process kNN queries for other fractional distance metrics.

In order to get more precise results, we propose a method called query-centric rehashing to search the base index and retrieve nearby objects. We further observe that during the processing of queries under different ℓ_p metrics, many common index entries are probed. This finding motivates us to optimize the processing of multiple queries under different ℓ_p metrics concurrently by sharing their I/Os.

We summarize the contributions as follows.

- We propose a novel method called LazyLSH to answer approximate nearest neighbor queries under multiple ℓ_p metrics. Compared to the costly naive method which builds an LSH index for every p value to cover all possible fractional distance metrics, LazyLSH maintains only a single copy of LSH index in the base space, significantly reducing the storage overhead.

- We give a theoretical proof that when certain conditions hold, locality-sensitive hash function can be extended to support the fractional distance metrics. This is the first work that gives a theoretical bound for the approximate kNN processing using LSH with the fractional distance metric.

- We propose two novel optimization methods, namely query-centric rehashing

Table 4.2: Table of Notations for Chapter 4

$\mathcal{D}$	database
d	dimensionality
$\vec{q}$	query point
$h_i^*(\cdot)$	based materialized hash function
$\vec{a}$	random vector in the hash function
b	offset in the hash function
c	approximation ratio
X	random variable
$\ell_p(\vec{o}, \vec{q})$	the ℓ_p distance between $\vec{o}$ and $\vec{q}$
p	the subscript used in the ℓ_p space or ℓ_p distance
δ	radius in the ℓ_p space
r	radius in the ℓ_1 space
$\delta^\perp$	lower bound of the ℓ_1 distance given $\ell_p = \delta$
$\delta^\top$	upper bound of the ℓ_1 distance given $\ell_p = \delta$
$B_p(\vec{q}, r)$	the ball of radius r centered at $\vec{q}$ in the ℓ_p space
η	the number of required hash functions
θ	the collision count threshold

and multi-query optimization, to improve the effectiveness and efficiency of performing queries.

- We experimentally verify the effectiveness and efficiency of our proposed LazyLSH using both synthetic and real datasets. Experimental results show that LazyLSH provides more accurate results for approximate kNN search under fractional distance metrics, and it can be used as a supervision to optimally choose the metric for different applications.

The rest of this chapter is organized as follows. Section 4.2 briefly reviews the preliminaries on LSH. Section 4.3 presents the technical details of the proposed LazyLSH method. Section 4.4 shows the processing of approximate range queries and approximate kNN queries. Then, we experimentally evaluate the proposed method in Section 4.5. Finally, we conclude this chapter in Section 4.6. For the ease of presentation, we summarize our notations in Table 4.2.

4.2 LazyLSH

In this section, we present LazyLSH as an efficient mechanism to process approximate kNN queries in different ℓ_p spaces. We begin with an overview, and then illustrate the technical details of LazyLSH.

4.2.1 Overview

In Chapter 2.3, we reviewed the existing methods of LSH. In previous work such as E2LSH and C2LSH, an LSH index is built for the ℓ_2 space, while Aggarwal et al. [2] showed that the fractional distance metrics ($0 < p < 1$) provide more meaningful results and improve the effectiveness of information retrieval algorithms. Extending techniques in E2LSH and C2LSH to support an arbitrary fractional distance metric is not a trivial task. We present LazyLSH, a novel approach that can process approximate nearest neighbor queries using different fractional distance metrics with a single LSH index.

LazyLSH is proposed based on the intuition that given $p, s > 0$, if two points are close in the ℓ_p space, then they are likely to be close in the ℓ_s space. Let $\epsilon = |p - s|$. The property holds with a higher probability for a smaller ϵ. This property can be extended as: the (r, cr, p_1, p_2)-sensitive hash function in the ℓ_p space is (δ, $c\delta$, p_1', p_2')-sensitive in the ℓ_s space if certain conditions hold. We will give a detailed theoretical analysis for this property later in this section.

Using this property, LazyLSH can transform the hash functions between different ℓ_p spaces. LSH families for the ℓ_1 and ℓ_2 metrics have been well studied [6, 132, 55]. As the ℓ_1 metric is closer to the fractional distance metrics, in LazyLSH, we materialize the LSH index in the ℓ_1 space as our base index. We generate η base hash functions $h_1^*(\cdot)$, ..., $h_\eta^*(\cdot)$. In particular, $h_i^*(\cdot)$ is constructed as:

$$h_i^*(\vec{v}) = \lfloor \frac{\vec{a_i}.\vec{v} + b_i^*}{r_0} \rfloor, \tag{4.1}$$

where a_i is a random vector whose each entry is drawn from the 1-stable (Cauchy) distribution and the other parameters are the same as the ones in Equation 2.6. Each base hash function h_i^* is $(1,\ c,\ p_1,\ p_2)$-sensitive, with $p_1 = p(1, r_0)$, $p_2 = p(c, r_0)$ as presented in Equation 2.3. How to select a proper η will be discussed later.

Using the materialized index, LazyLSH can answer the approximate kNN queries, as stated as $\mathcal{N}_p(\vec{q}, k, c)$ in Definition 2.3, with probabilistic guarantees. For the ease of presentation, we first show our observation that a $(r,\ cr,\ p_1,\ p_2)$-sensitive hash function in the ℓ_p space is $(\delta,\ c\delta,\ p_1',\ p_2')$-sensitive in another ℓ_s space in this section. Then we illustrate how the LazyLSH method answers $\mathcal{R}_p(\vec{q}, r, c)$ and $\mathcal{N}_p(\vec{q}, k, c)$ queries in Section 4.3.

4.2.2 LSH in an ℓ_p Space

If there exists a point $\vec{o} \in B_p(\vec{q}, \delta)$, the $\mathcal{R}_p(\vec{q},\ \delta,\ c)$ query returns a point $\vec{o'}$ if $\vec{o'} \in B_p(\vec{q}, c\delta)$. We observe that $B_p(\vec{q}, \delta)$ and $B_1(\vec{q}, r)$ share a lot of common areas if r is carefully tuned for δ. Figure 4.2 plots a $B_1(\vec{q}, r)$ ball in blue and a $B_p(\vec{q}, \delta)$ ball in red, where $0 < p < 1$. The shadow region represents the intersection of $B_1(\vec{q}, r)$ and $B_p(\vec{q}, \delta)$ which takes over a large portion of $B_p(\vec{q}, \delta)$. This observation motivates us to use a ball $B_1(\vec{q}, r)$ in the ℓ_1 space to approximate $B_p(\vec{q}, \delta)$ in the ℓ_p space for the query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$.

Given two points $\vec{q},\ \vec{o} \in \mathbb{R}^d$, with $\ell_p(\vec{q}, \vec{o}) = \delta$, we can compute the distance lower bound and upper bound in the ℓ_1 space, denoted as $\delta^\perp$ and $\delta^\top$ respectively. Figure 4.3 shows the geometric interpretations of $\delta^\perp$ and $\delta^\top$ for $0 < p < 1$ and $p > 1$. The values of $\delta^\perp$ and $\delta^\top$ are computed as:

$$\delta^\perp = \begin{cases} \frac{d \times \delta}{\sqrt[p]{d}} & \text{if } 0 < p < 1 \\ \delta & \text{if } p \geqslant 1 \end{cases} \qquad \delta^\top = \begin{cases} \delta & \text{if } 0 < p < 1 \\ \frac{d \times \delta}{\sqrt[p]{d}} & \text{if } p \geqslant 1 \end{cases} \tag{4.2}$$

Our goal is to use an ℓ_1 ball $B_1(\vec{q}, r)$ to approximate the ℓ_p ball $B_p(\vec{q}, \delta)$ specified by the query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$. The radius r significantly affects the search performance. If

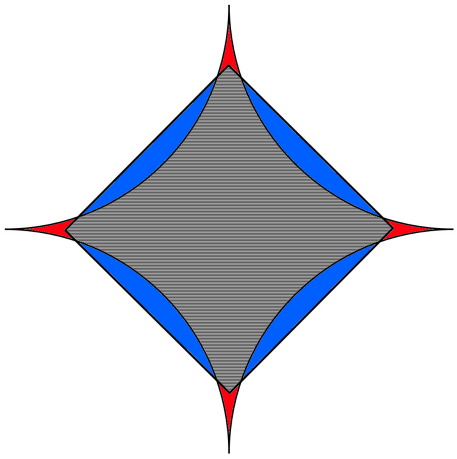
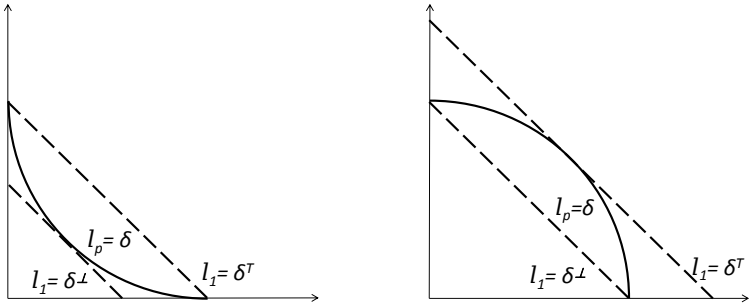


Figure 4.2: Use $B_1(\vec{q}, r)$ to approximate $B_p(\vec{q}, \delta)$ (Best viewed in color)



(a) $0 < p < 1$ (b) $p > 1$

Figure 4.3: Bounds of ℓ_p distance

$r < \delta^{\perp}$, we may fail to retrieve many candidate results, leading to high false negatives. On the other hand, if $r > \delta^{\top}$, many irrelevant objects are retrieved, generating many false positives. Therefore, a proper r should be chosen in the range of $[\delta^{\perp}, \delta^{\top}]$ for $B_1(\vec{q}, r)$ to approximate $B_p(\vec{q}, \delta)$.

Given a query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$ and a based $(1, c, p_1, p_2)$-sensitive hash function $h_i^*(\cdot)$, we modify $h_i^*(\cdot)$ as follows:

$$h_i^r(\vec{v}) = \lfloor \frac{\vec{a_i}.\vec{v} + b_i^*}{r_0 r} \rfloor, \tag{4.3}$$

where r is in $[\delta^{\perp}, \delta^{\top}]$ as discussed above. It is easy to see that $h_i^r(\cdot)$ is $(r,\ cr,\ p(r, r_0 r),\ p(cr, r_0 r))$-sensitive in the ℓ_1 space, where $p(\cdot, \cdot)$ is defined in Equation 2.3. We

further observe the following lemma.

Lemma 4.1. *Let* $p(s,r) = \int_0^r \frac{1}{s} f_p(\frac{t}{s})(1-\frac{t}{r})dt$ *as shown in Equation 2.3. For any real number* $c > 0$, $p(s,r) = p(cs,cr)$.

Proof. We know:

$$p(cs,cr) = \int_0^{cr} \frac{1}{cs} f_p(\frac{t}{cs})(1-\frac{t}{cr})dt$$

Let

$$x = t/c,$$

Then we get:

$$\begin{aligned} p(cs,cr) &= \int_0^r c\frac{1}{cs} f_p(\frac{cx}{cs})(1-\frac{cx}{cr})dx \\ &= \int_0^r \frac{1}{s} f_p(\frac{x}{s})(1-\frac{x}{r})dx = p(s,r) \quad \square \end{aligned}$$

By Lemma 4.1, we get $p(r, r_0 r) = p(1, r_0) = p_1$ and $p(cr, r_0 r) = p(c, r_0) = p_2$, where p_1 and p_2 are the same p_1 and p_2 defined in the base materialized hash function $h_i^*(\cdot)$.

Recall that our LSH index is built in the ℓ_1 space. The LSH family guarantees that close points in the ℓ_1 space are likely to be hashed into the same bucket. We are interested in whether an LSH function in the ℓ_1 space is still locality-sensitive in another ℓ_p space. Given a (r, cr, p_1, p_2)-sensitive hash function $h_i^r(\cdot)$ in the ℓ_1 space, we define the following events for any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$:

$$\begin{array}{lll} e_1 : h_i^r(\vec{o}) = h_i^r(\vec{q}). & & \\ e_2 : \ell_p(\vec{o},\vec{q}) \leqslant \delta. & e_3 : \ell_p(\vec{o},\vec{q}) > c\delta. & \text{where } c > 1 \\ e_4 : \ell_1(\vec{o},\vec{q}) \leqslant r. & e_5 : \ell_1(\vec{o},\vec{q}) > cr. & \text{where } c > 1 \end{array}$$

To verify whether the modified hash function $h_i^r(\cdot)$ is locality-sensitive in the ℓ_p space, we need to calculate the probability of e_1 given e_2 and the probability of e_1

given e_3, i.e. $Pr(e_1|e_2)$ and $Pr(e_1|e_3)$ respectively. Let e^c represent the complementary of event e. By Bayes' Theorem, we compute a lower bound of $Pr(e_1|e_2)$:

$$\begin{aligned}
Pr(e_1|e_2) &= Pr(e_1 \wedge e_4|e_2) + Pr(e_1 \wedge e_4^c|e_2) \\
&= Pr(e_4|e_2)Pr(e_1|e_4 \wedge e_2) + Pr(e_4^c|e_2)Pr(e_1|e_4^c \wedge e_2) \\
&\text{(Note: } Pr(e_1|e_4 \wedge e_2) \geqslant Pr(e_1|e_4) \geqslant p_1, \text{ and} \\
&\ell_p(\vec{o}, \vec{q}) \leqslant \delta \text{ implies } \ell_1(\vec{o}, \vec{q}) \leqslant \delta^\top \text{)} \\
&\geqslant Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(\delta^\top, r_0 r) \\
&= Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(1, \frac{r_0 r}{\delta^\top}) \qquad (4.4)
\end{aligned}$$

This is because $p(s, r)$ is monotonically decreasing with s when r is fixed. We infer $p(1, \frac{r_0 r}{\delta^\top}) \leqslant p_1$ from $\delta^\perp \leqslant r \leqslant \delta^\top$. For simplicity, we use p_1' to denote $Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(1, \frac{r_0 r}{\delta^\top})$. Consequently, we get $p_1' \leqslant p_1$. Similarly, we compute an upper bound of $Pr(e_1|e_3)$:

$$\begin{aligned}
Pr(e_1|e_3) =& Pr(e_1 \wedge e_5|e_3) + Pr(e_1 \wedge e_5^c|e_3) \\
=& Pr(e_5|e_3)Pr(e_1|e_5 \wedge e_3) + Pr(e_5^c|e_3)Pr(e_1|e_5^c \wedge e_3) \\
&\text{(Note: } Pr(e_1|e_5 \wedge e_3) \leqslant Pr(e_1|e_5) \leqslant p_2, \text{ and} \\
&\ell_p(\vec{o}, \vec{q}) > c\delta \text{ implies } \ell_1(\vec{o}, \vec{q}) > c\delta^\perp \text{)} \\
\leqslant& Pr(e_5|e_3)p_2 + (1 - Pr(e_5|e_3))p(c\delta^\perp, r_0 r) \\
=& Pr(e_5|e_3)p_2 + (1 - Pr(e_5|e_3))p(c, r_0 r/\delta^\perp) \\
\leqslant& \begin{cases} p_2 & \text{if } p_2 \geqslant p(c, r_0 r/\delta^\perp), \text{ i.e. } r \leqslant \delta^\perp \\ p(c, r_0 r/\delta^\perp) & \text{otherwise} \end{cases} \\
&\text{(Note: we set } \delta^\perp \leqslant r \leqslant \delta^\top \text{)} \\
=& p(c, r_0 r/\delta^\perp) \qquad (4.5)
\end{aligned}$$

Then, we use p_2' to denote $p(c, r_0 r/\delta^{\perp})$. Based on the above two inequalities, we have the following theorem:

Theorem 4.1. *Given an* $(r,\ cr,\ p_1,\ p_2)$*-sensitive hash function* $h_i^r(\cdot)$ *in the* ℓ_1 *space, we find the following two conditions hold for a distance threshold* δ *in another* ℓ_p *space, where* $\delta^{\perp} \leqslant r \leqslant \delta^{\top}$*:*

(1) if $\ell_p(\vec{o}, \vec{q}) \leqslant \delta$*, then* $Pr[h_i^r(\vec{o}) = h_i^r(\vec{q})] \geqslant p_1'$*,*

(2) if $\ell_p(\vec{o}, \vec{q}) > c\delta$*, then* $Pr[h_i^r(\vec{o}) = h_i^r(\vec{q})] < p_2'$*,*

where p_1' *and* p_2' *are stated as stated above.*

$$p_1' = Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(1, \frac{r_0 r}{\delta^{\top}})$$

$$p_2' = p(c, r_0 r/\delta^{\perp})$$

4.2.3 Computing Internal Parameters

To ensure the correctness of LazyLSH, two parameters are needed to compute. One is the radius r of an ℓ_1 ball $B_1(\vec{q}, r)$ to approximate the ℓ_p ball $B_p(\vec{q}, \delta)$ for query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$. The other is the number of required hash functions to be built. Then we present how to compute them.

Recall the definition of LSH, we must have $p_1' > p_2'$ so that the locality-sensitive hash function $h_i^r(\cdot)$ is useful in the ℓ_p space. By substituting p_1' and p_2', we have $p_2' = p(c, r_0 r/\delta^{\perp}) = p(c\delta^{\perp}/r, r_0) < p_1' \leqslant p_1 = p(1, r_0)$. Note that $p(s, r)$ is monotonically decreasing with s when r is fixed. Thus, we get $c\delta^{\perp}/r > 1$ as a necessary condition, which infers $r < c\delta^{\perp}$. Besides, we have $\delta^{\perp} \leqslant r \leqslant \delta^{\top}$ as we explained before. In summary, r must be chosen properly in the range of $[\delta^{\perp}, \min(\delta^{\top}, c\delta^{\perp})]$ in order for $p_1' > p_2'$ to hold. In details, r is a parameter for the functions of p_1' and p_2'. p_2' can be simply computed using Equation 2.4 when we build the base LSH index in the ℓ_1 space, while computing p_1' is a nontrivial task. We begin with a lemma on the conditional probability.

Lemma 4.2. *$Pr(\ell_s(\vec{x},\vec{y}) \leqslant r | \ell_p(\vec{x},\vec{y}) \leqslant \delta) = Pr(\ell_s(\vec{u},\vec{v}) \leqslant cr | \ell_p(\vec{u},\vec{v}) \leqslant c\delta)$ for any $s, p, c > 0$.*

Proof. For any $\vec{x}, \vec{y} \in \mathbb{R}^d$, let $\vec{u} = c\vec{x}$ and $\vec{v} = c\vec{y}$. Then, we have

$$\begin{aligned}\ell_s(\vec{u},\vec{v}) =& (\sum_{i=1}^{d}(u_i - v_i)^s)^{1/s} \\ =& (\sum_{i=1}^{d}(cx_i - cy_i)^s)^{1/s} \\ =& c\ \ell_s(\vec{x},\vec{y})\end{aligned}$$

Therefore, we get $\ell_s(\vec{x},\vec{y}) \leqslant r \iff \ell_s(\vec{u},\vec{v}) \leqslant cr$. Similarly, $\ell_p(\vec{x},\vec{y}) \leqslant \delta \iff \ell_p(\vec{u},\vec{v}) \leqslant c\delta$. Note $\vec{u}$ and $\vec{x}$, $\vec{v}$ and $\vec{y}$ are one-to-one corresponding. Thus, we get

$$Pr(\ell_s(\vec{x},\vec{y}) \leqslant r | \ell_p(\vec{x},\vec{y}) \leqslant \delta) = Pr(\ell_s(\vec{u},\vec{v}) \leqslant cr | \ell_p(\vec{u},\vec{v}) \leqslant c\delta)$$

□

Based on Lemma 4.2, we can reduce the problem of calculating $Pr(e_4|e_2)$ to the computation of $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r \mid \ell_p(\vec{o},\vec{q}) \leqslant 1)$, where $\delta = 1$ and $\delta^{\perp} \leqslant r < \min(\delta^{\top}, c\delta^{\perp})$. $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$ can be computed as follows:

$$Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1) = \frac{Vol(B_1(\vec{q},r) \bigcap B_p(\vec{q},1))}{Vol(B_p(\vec{q},1))}, \tag{4.6}$$

where $Vol(\cdot)$ outputs the volume of a given shape.

The challenge, however, is that computing the volume of $(B_1(\vec{q},r) \bigcap B_p(\vec{q},1))$ for a random p and r exactly is very expensive if not impossible. Alternatively, we use the Monte Carlo method [19] to estimate $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$. Basically, this estimation is done by randomly sampling points in $B_p(\vec{q},1)$ and then calculating the percentage of the sampled points in $B_1(\vec{q},r)$. Suppose we randomly sample n

Algorithm 2: Sampling a point in $B_p(\vec{o}, 1)$ [24]

input : dimensionality d, ℓ_p space
output: a random point $\vec{y}$ in $B_p(\vec{o}, 1)$, where $\vec{o}$ is the origin

1 Generate d independent random real scalars $\xi_i \sim \tilde{G}(\frac{1}{p}, p)$;
2 Construct a vector $\vec{x} \in \mathbb{R}^d$ of components $x_i = s_i \xi_i$, where s_i is an independent random sign;
3 Construct $z = w^{1/d}$, where w is a random variable uniformly distributed in the interval [0, 1] ;
4 **return** $\vec{y} = z \frac{\vec{x}}{\ell_p(\vec{x}, \vec{o})}$;

points in $B_p(\vec{q}, 1)$, and find m points are in $B_1(\vec{q}, r)$ as well. $\frac{m}{n}$ is roughly equal to $Pr(\ell_1(\vec{o}, \vec{q}) \leqslant r | \ell_p(\vec{o}, \vec{q}) \leqslant 1)$ if the number of samples is large enough.

$$Pr(\ell_1(\vec{o}, \vec{q}) \leqslant r | \ell_p(\vec{o}, \vec{q}) \leqslant 1) \approx \frac{m}{n}, \tag{4.7}$$

Note that the location of the center does not affect the probability. As a result, we sample points in $B_p(\vec{o}, 1)$, where $\vec{o}$ represents the origin. Given a d-dimensional space and a p value, we can randomly sample points in $B_p(\vec{o}, 1)$ using Algorithm 2 [24]. In line 1, ξ_i is a random variable generated from a Generalized Gamma density function.

Definition 4.1 (Generalized Gamma density [83])**.** *A random variable $x \in \mathbb{R}$ is generalized gamma distributed with three parameters $\lambda > 0$, $\beta > 0$ and $\upsilon > 0$, denoted as $x \sim \tilde{G}(\lambda, \beta, \upsilon)$, when x has density function:*

$$f(x) = \frac{\upsilon / \lambda^{\beta}}{\Gamma(\beta / \upsilon)} x^{\beta - 1} e^{-(x/\lambda)^{\upsilon}}, x \geqslant 0. \tag{4.8}$$

Let $\lambda = 1$, $a = \beta / \upsilon$ and $c = \upsilon$, we can get a generalized gamma distribution with two parameters $a > 0$ and $c > 0$, denoted as $x \sim \tilde{G}(a, c)$. The density function is:

$$f(x) = \frac{c}{\Gamma(a)} x^{ca - 1} e^{-x^c}, x \geqslant 0. \tag{4.9}$$

Algorithm 3: Calculating $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$

input : dimensionality d, ℓ_p space, the number of sample points n, the number of buckets b
output: an array p storing $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$ with different values of r

1 Initialize an array r of b dimensions recording the radii, $r[i] = \delta^{\perp} + (i+1) \times \frac{\min(\delta^{\top}, c\delta^{\perp}) - \delta^{\perp}}{b}$ for $0 \leqslant i < b$;
2 Initialize an array c of b dimensions to record the number of points in $B_1(\vec{o}, r)$, $c[i] \leftarrow 0$ for $0 \leqslant i < b$;
3 **for** $k \leftarrow 0$ **to** $n-1$ **do**
4 $\vec{v} \leftarrow$ a random point in $B_p(\vec{o}, 1)$;
5 Compute $\ell_1(\vec{o}, \vec{v})$; `/* o is the origin */`
6 Find the minimal index j such that $r[j] \geqslant \ell_1(\vec{o}, \vec{v})$;
7 **for** $i \leftarrow j$ **to** $b-1$ **do**
8 $c[i] \leftarrow c[i] + 1$;
9 **for** $i \leftarrow 0$ **to** $b-1$ **do**
10 $p[i] \leftarrow \frac{c[i]}{n}$;
11 **return** p;

The formula involves the gamma function $\Gamma(t)$ $(t > 0)$, which is computed as:

$$\Gamma(t) = \int_0^{\infty} x^{t-1} e^{-x} dx, t > 0. \tag{4.10}$$

We can obtain a random Generalized Gamma variable $x \sim \tilde{G}(a, c)$ from a Gamma random variable $z \sim G(a, 1)$ as $x = z^{1/c}$ [83]. As a result, we can efficiently generate sample points in $B_p(\vec{o}, 1)$ using Algorithm 2.

Then we compute $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$ w.r.t. different values of r. As $\delta^{\perp} \leqslant r \leqslant \min(\delta^{\top}, c\delta^{\perp})$, we divide $[\delta^{\perp}, \min(\delta^{\top}, c\delta^{\perp})]$ into b buckets. So the bucket length ϕ is $(\min(\delta^{\top}, c\delta^{\perp}) - \delta^{\perp})/b$. Afterwards, we set r to $(\delta^{\perp} + \phi)$, $(\delta^{\perp} + 2\phi)$, ..., $\min(\delta^{\top}, c\delta^{\perp})$ respectively and compute $Pr(\ell_1(\vec{o},\vec{q}) \leqslant r | \ell_p(\vec{o},\vec{q}) \leqslant 1)$, as described in Algorithm 3.

We initialize a counter to record the number of sample points that are located in $B_1(\vec{o}, r)$ for each radius r (line 2). Then, we randomly sample n points in $B_p(\vec{o}, 1)$. For each sampled point $\vec{v}$, we calculate its distance $\ell_1(\vec{v}, \vec{o})$ to $\vec{o}$ in the ℓ_1 space.

For all the radii r greater than $\ell_1(\vec{v}, \vec{o})$, we know that the sample point $\vec{v}$ is inside $B_1(\vec{o}, r)$. Then we add one to the corresponding counters (lines 6-8). After sampling n points, we output $Pr(\ell_1(\vec{o}, \vec{q}) \leqslant r | \ell_p(\vec{o}, \vec{q}) \leqslant 1)$ for different values of r (lines 9-11). Algorithm 2 is an offline process. This approximation becomes more accurate if we sample more points and maintain more buckets. In the experiments, we set the number of sample points n to 1,000,000 and the number of buckets b to 1000.

Once we get the value of $Pr(\ell_1(\vec{o}, \vec{q}) \leqslant r | \ell_p(\vec{o}, \vec{q}) \leqslant 1)$, which is equal to $Pr(e_4|e_2)$, we can compute p'_1 and p'_2 as shown in Equations 4.4 and 4.5. In particular, $p(\cdot, \cdot)$ is computed as the one in Equation 2.4, because we build the base index in the ℓ_1 space. Figure 4.4 plots the values of p'_1 and p'_2 w.r.t. r for $\ell_{0.5}$ in $\mathbb{R}^{128}$ when the approximate ratio c is set to 2. The x axis represents the ratio of r to $\delta_\perp$, i.e. $(\frac{r}{\delta_\perp})$. As can be seen in the figure, p'_2 increases smoothly. In contrast, p'_1 increases slowly at the beginning. When the ratio reaches 1.4, p'_1 increases dramatically. When the ratio reaches around 1.55, p'_1 exceeds p'_2 and grows slowly at the end. We are interested in the cases when $p'_1 > p'_2$.

Recall that a (1, c, p_1, p_2)-sensitive hash function requires $p_1 > p_2$. In addition, the number of base hash functions η must be set to a certain value to ensure the correctness of the algorithm, which is related to $(p_1 - p_2)$ as shown in Equation 2.8 [55]. Equation 2.8 shows that the greater $(p_1 - p_2)$ is, the less base hash functions are required, resulting in less storage overhead. Therefore, we choose an optimal radius r, denoted as $\hat{r}$, by maximizing $(p'_1 - p'_2)$.

$$\hat{r} = \arg\max_r \ (p'_1 - p'_2) \tag{4.11}$$

For different ℓ_p spaces, the optimal $\hat{r}$ varies. We precompute and save the values of $\hat{r}$ for different ℓ_p spaces, which is used when processing a query. Besides, we store the values of p'_1 and p'_2 when $r = \hat{r}$ which are denoted as $\hat{p}'_1$ and $\hat{p}'_2$ respectively. They are used to compute the number of required hash functions η when building the index

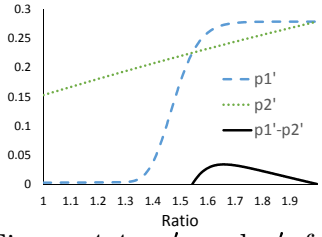


Figure 4.4: p'_1 and p'_2 for $\ell_{0.5}$ in $\mathbb{R}^{128}$, $c = 2$

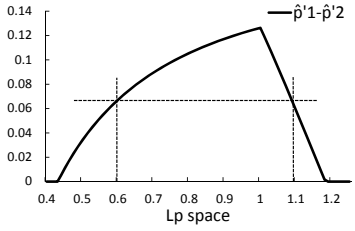


Figure 4.5: $(\hat{p}'_1 - \hat{p}'_2)$ w.r.t the ℓ_p spaces in $\mathbb{R}^{128}$, $c = 2$

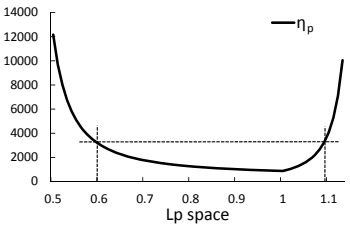


Figure 4.6: Number of functions required in $\mathbb{R}^{128}$

and the collision count threshold θ in processing a query as shown in Equations 2.8 and 2.9. Figure 4.5 plots $(\hat{p}'_1 - \hat{p}'_2)$ w.r.t. different ℓ_p spaces in $\mathbb{R}^{128}$, where the approximate ratio c is set to 2. When $p < 1$, $(\hat{p}'_1 - \hat{p}'_2)$ drops when p decreases. When $p < 0.44$, $\hat{p}'_1$ is always smaller than $\hat{p}'_2$, which indicates that the hash function built in the ℓ_1 space is no longer locality-sensitive in the corresponding ℓ_p space when $c = 2$. When $p > 1$, $(\hat{p}'_1 - \hat{p}'_2)$ drops significantly when p increases. When $p > 1.18$, $\hat{p}'_1$ is always smaller than $\hat{p}'_2$.

Next, we calculate the number of required hash functions η for different ℓ_p spaces. We use η_p to denote the number of required hash functions to support queries in the ℓ_p space, and it is computed as

$$\eta_p = \lceil \frac{\ln \frac{1}{\varepsilon}}{2(\hat{p}'_1 - \hat{p}'_2)^2}(1+z)^2 \rceil, \text{ where } z = \sqrt{\frac{\ln \frac{2}{\beta}}{\ln \frac{1}{\varepsilon}}} \tag{4.12}$$

where ε and β are the same as the ones defined in Equation 2.8. Figure 4.6 plots η_p w.r.t. different ℓ_p spaces in $\mathbb{R}^{128}$, where $c = 2$, $\varepsilon = 0.01$ and $\beta = 0.0001$. When $p < 1$, η_p increases when p decreases, because η_p is inversely proportional to $(\hat{p}'_1 - \hat{p}'_2)$. Suppose we want to support a query $\mathcal{R}_{0.6}(\vec{q}, \delta, c)$ in the $\ell_{0.6}$ space. Correspondingly, at least $\eta_{0.6}$ hash functions are needed to be materialized for the base index. Suppose we materialize $\eta_{0.6}$ hash functions as the base index. Using $\eta_{0.6}$ base hash functions, queries $\mathcal{R}_p(\vec{q}, r, c)$ can be answered in the ℓ_p spaces where $\eta_p \leqslant \eta_{0.6}$, as shown by the dashed line $(0.6 \leqslant p \leqslant 1.1)$ in Figure 4.6.

4.3 Query Processing

In this section, we discuss how to leverage LazyLSH to process approximate range queries ($\mathcal{R}_p(\vec{q}, \delta, c)$) and nearest neighbor queries ($\mathcal{N}_p(\vec{q}, k, c)$) in different ℓ_p spaces.

4.3.1 Processing $\mathcal{R}_p(\vec{q}, \delta, c)$

Equation 4.12 indicates that we can find η_s and $\eta_{s'}$ with $s < s'$ and $\eta_s = \eta_{s'}$. Namely, two spaces share the same η value. This idea is also shown in Figure 4.5, where we get the same value of $|\hat{p}'_1 - \hat{p}'_2|$ in $\ell_{0.6}$ and $\ell_{1.1}$. Suppose we have materialized η_s hash functions as the base index, where $0 < s < 1$. $\mathcal{R}_p(\vec{q}, \delta, c)$ queries can be answered in a series of ℓ_p spaces using the base index, where $s \leqslant p \leqslant s'$.

We use the $B_1(\vec{q}, \hat{r}\delta)$ ball in the ℓ_1 space to approximate the $B_p(\vec{q}, \delta)$ ball in the ℓ_p space as described in the previous section. We also precompute the corresponding $\hat{p}'_1$ and $\hat{p}'_2$ for the query ℓ_p space. To answer an $\mathcal{R}_p(\vec{q}, \delta, c)$ query, we modify the base hash function $h_i^*(\cdot)$ as $h_i^{\hat{r}\delta}(\cdot)$

$$h_i^{\hat{r}\delta}(\vec{v}) = \lfloor \frac{h_i^*(\vec{v}) - (h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor) + \lfloor \frac{3\hat{r}\delta}{2} \rfloor}{\hat{r}\delta} \rfloor \tag{4.13}$$

$(h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor)$ can be viewed as a random integer as $\vec{q}$ is not known beforehand. Thus, $(\lfloor \frac{3\hat{r}\delta}{2} \rfloor - (h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor))$ is a random positive integer, and $h_i^{\hat{r}\delta}(\cdot)$ is $(\hat{r}\delta, c\hat{r}\delta, p_1, p_2)$-sensitive in the ℓ_1 space. Based on Theorem 4.1, we know that $h_i^{\hat{r}\delta}(\cdot)$ is also $(\delta, c\delta, \hat{p}'_1, \hat{p}'_2)$-sensitive in the ℓ_p space.

Given a query $\mathcal{R}_p(\vec{q}, \delta, c)$ and the base hash functions, if an object collides with $\vec{q}$ more than θ_p times, the object is considered as a candidate. In particular, θ_p is defined as:

$$\theta_p = \frac{z\hat{p}'_1 + \hat{p}'_2}{1 + z}\eta_s, \tag{4.14}$$

The $\mathcal{R}_p(\vec{q}, \delta, c)$ query is processed by retrieving the objects that are hashed to the same bucket as $\vec{q}$ for $h_i^{\delta\hat{r}}(\cdot)$. We adopt the virtual rehashing method [55]. If a point

Algorithm 4: Answering $\mathcal{R}_p(\vec{q}, \delta, c)$

```
T ← a hash table to record the collision count for each database object ;
C ← a list to record the candidates ;
r̂ ← the radius of B_1(q⃗, r̂) to approximate B_p(q⃗, 1) ;
for i ← 0 to η_s − 1 do
    Compute h_i^{δr̂}(q⃗) ;
    Read the IDs that are hashed in the range of [ h_i^*(q⃗) − ⌊δr̂/2⌋, h_i^*(q⃗) + ⌊δr̂/2⌋ ]
    for h_i^*(·) ;
    foreach object v⃗ ∈ IDs do
        T[v⃗] ← T[v⃗] + 1 ;
        if (T[v⃗] > θ_p) ∧ (v⃗ ∉ C) then
            C ← C ⋃ {v⃗} ;
            if ℓ_p(v⃗, q⃗) < cδ then
                return v⃗ ;
            if |C| > β|D| then
                return NULL ;
```

$\vec{v}$ has the same hash value as the query, i.e. $h_i^{\delta\hat{r}}(\vec{v}) = h_i^{\delta\hat{r}}(\vec{q})$, we can get the value range of $h_i^*(\vec{v})$ by expanding the equation.

$$h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2}\rfloor \leqslant h_i^*(\vec{v}) \leqslant h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2}\rfloor \tag{4.15}$$

From the above equation, we observe that the points hashed into the range of $[\, h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2}\rfloor, h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2}\rfloor\,]$ in $h_i^*(\cdot)$ will collide with the query point which is mapped to the same bucket $h_i^{\delta\hat{r}}(\vec{q})$. We can see that the center of this range is the query point. Thus, we refer to the proposed hash function $h_i^{\hat{r}\delta}(\cdot)$ as the *query-centric rehashing* function.

Figure 4.7 presents the advantage of the proposed *query-centric rehashing* function. Suppose $\vec{q}$ is the query point. The first line represents the based hash function $h^*(\vec{q})$, where points $\vec{v}$, $\vec{q}$ and $\vec{o}$ are hashed to buckets 8, 9 and 13 respectively. The solid rectangle represents our proposed *query-centric rehashing* function for radius $r = 3, 9$, whereas the dashed rectangle represents the original rehashing function in

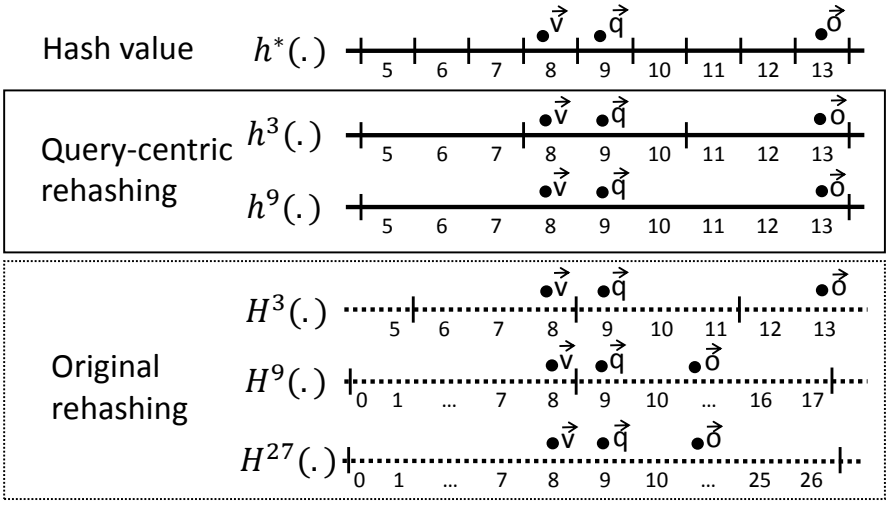


Figure 4.7: Query-centric rehashing

Equation 2.7. For the *query-centric rehashing* function $h^r(\cdot)$, $\vec{q}$ and $\vec{v}$ are hashed to the same bucket when $r = 3$, while $\vec{q}$ and $\vec{o}$ are hashed to the same bucket when $r = 9$. In contrast, for $H^r(\cdot)$, $\vec{q}$ and $\vec{o}$ are hashed to the same bucket when $r = 9$, while $\vec{q}$ and $\vec{v}$ are hashed to the same bucket when $r = 27$. We notice that the query point $\vec{q}$ is actually closer to $\vec{v}$ compared with $\vec{o}$, and thus $\vec{q}$ should collide with $\vec{v}$ in the same bucket first. Our proposed *query-centric rehashing* function can hold this property while $H^r(\cdot)$ cannot. In particular, $H^r(\cdot)$ might perform poorly in some cases, for example, when a query point is hashed to the bucket whose id is a multiple of the radius.

Algorithm 4 shows the pseudo code of processing an $\mathcal{R}_p(\vec{q}, \delta, c)$ query. An object with collision count larger than θ_p is considered as a candidate and its real distance to the query is computed. The algorithm stops until $\beta|\mathcal{D}|$ candidates are found. The algorithm is sound if the following two properties hold with a constant probability.

1. $\mathcal{P}_1'$: If $\vec{v} \in B_p(\vec{q}, \delta)$, then the number of $\vec{v}$'s collision with the query $\vec{q}$ is at least θ_p.

2. $\mathcal{P}_2'$: The total number of false positives is smaller than $\beta|\mathcal{D}|$, where $|\mathcal{D}|$ is the

Algorithm 5: Answering $\mathcal{N}_p(\vec{q}, k, c)$

$T \leftarrow$ a hash table to record the collision count for each database object ;
$C \leftarrow$ a list to record the candidates ;
$\hat{r} \leftarrow$ the radius of $B_1(\vec{q}, \hat{r})$ to approximate $B_p(\vec{q}, 1)$;
$\delta \leftarrow \frac{1}{\hat{r}}$;
while *TRUE* **do**
  for $i \leftarrow 0$ **to** $\eta_s - 1$ **do**
    if $\delta = \frac{1}{\hat{r}}$ **then**
      $IDs \leftarrow$ the points that are hashed to $h_i^*(\vec{q})$;
    else
      $IDs \leftarrow$ the points that are hashed in $[\ h_i^*(\vec{q}) - \lfloor \frac{\delta \hat{r}}{2} \rfloor, h_i^*(\vec{q}) - \lfloor \frac{\delta \hat{r}}{2c} \rfloor - 1\ ]$ or $[\ h_i^*(\vec{q}) + \lfloor \frac{\delta \hat{r}}{2c} \rfloor + 1, h_i^*(\vec{q}) + \lfloor \frac{\delta \hat{r}}{2} \rfloor\ ]$ for $h_i^*(\cdot)$;
    foreach *object* $\vec{v} \in IDs$ **do**
      $T[\vec{v}] \leftarrow T[\vec{v}] + 1$;
      if $(T[\vec{v}] > \theta_p) \wedge (\vec{v} \notin C)$ **then**
        $C \leftarrow C \bigcup \{\vec{v}\}$;
        if $(|\{\vec{o} | \vec{o} \in C \wedge \ell_p(\vec{o}, \vec{q}) < c\delta\}| \geqslant k) \vee (|C| > k + \beta|\mathcal{D}|)$ **then**
          return the kNNs in C with the smallest ℓ_p distance to $\vec{q}$;
  $\delta \leftarrow \delta * c$;

cardinality of a database $\mathcal{D}$.

Corollary 4.1 guarantees the correctness of the algorithm.

Corollary 4.1. *Given a error probability ε and a false positive rate β, if η_p is set as the one in Equation 4.12, we have $Pr(\mathcal{P}_1') \geqslant (1 - \varepsilon)$ and $Pr(\mathcal{P}_2') \geqslant 1/2$*

Proof. This is a corollary of Lemma 1 in [55]. We refer readers to [55] for more details. □

4.3.2 Processing $\mathcal{N}_p(\vec{q}, k, c)$

The nearest neighbor query can be processed in a similar way. Algorithm 5 presents the general idea which can be viewed as the processing of a set of $\mathcal{R}_p(\vec{q}, \delta, c)$ queries with increasing radii. We initialize the starting radius to be $\frac{1}{\hat{r}}$ and issue an $\mathcal{R}_p(\vec{q}, \delta, c)$

query. If not enough results are found, we increase the radius to be $c\delta$ and issue a new range query. The process continues until enough results are returned.

We iteratively retrieve the points that are hashed into the range of $[\ h_i^*(\vec{q}) - \lfloor \frac{\delta \hat{r}}{2} \rfloor$, $h_i^*(\vec{q}) + \lfloor \frac{\delta \hat{r}}{2} \rfloor\ ]$ for $h_i^*(\cdot)$, because they collide with the query point for the modified hash function. However, as we have visited the points that are hashed in $[\ h_i^*(\vec{q}) - \lfloor \frac{\delta \hat{r}}{2c} \rfloor$, $h_i^*(\vec{q}) + \lfloor \frac{\delta \hat{r}}{2c} \rfloor\ ]$ in the last iteration when the radius is (δ/c), we skip those IDs (line 10).

The algorithm stops if (1) we have obtained k candidates whose ℓ_p distance to q is smaller than $c\delta$, or (2) we have found more than $k + \beta|\mathcal{D}|$ candidates with collision count larger than θ_p (lines 15-16). The stop conditions are defined based on $\mathcal{P}_1'$ and $\mathcal{P}_2'$ respectively. In particular, $\mathcal{P}_1'$ guarantees that a candidate will be found definitely, and $\mathcal{P}_2'$ ensures that there are no more than $\beta|\mathcal{D}|$ false positives. Therefore, we can stop the algorithm early and return the approximate kNNs of q in the query ℓ_p space.

4.3.3 Multi-query Optimization

For the processing of queries under different ℓ_p metrics, the only difference is that the algorithm requires a different collision threshold for an object to become a candidate. For fractional distance metrics, the smaller the p is, the larger the collision threshold is. This requires that more index entries need to be retrieved for the fractional distance metrics with a smaller p. The good news is that the index entries for a larger collision threshold (a smaller p) cover the index entries for a smaller collision threshold (a larger p), which means that no additional sequential I/O is required when we process queries in different ℓ_p spaces simultaneously for the same query point.

This finding motivates us to perform multiple queries under different ℓ_p metrics concurrently by sharing their I/Os. For example, suppose we need to answer queries $\mathcal{N}_p(\vec{q}, k, c)$ for p = 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. We can group them and answer

them simultaneously. The I/O cost of processing these multiple queries is roughly the same as the I/O cost of processing a single $\mathcal{N}_{0.5}(\vec{q}, k, c)$ query with additional random I/Os for retrieving candidates of other ℓ_p metrics.

4.3.4 Extending LazyLSH to Other Existing Methods

LazyLSH is orthogonal to most previous work on LSH. LazyLSH targets at answering approximate nearest neighbor queries in different ℓ_p spaces using a single index. The same index structure of C2LSH is used to process approximate nearest neighbor queries. Besides C2LSH, LazyLSH can also adopt other index structures. For instance, LazyLSH can be easily combined with the E2LSH structure. The difference is how to choose the optimal value of $\hat{r}$ in Equation 4.11. To fit the E2LSH structure and its variants [42, 90], we choose $\hat{r}$ with a different optimization method. As stated in [42], $\rho = \frac{\ln 1/p'_1}{\ln 1/p'_2}$ is a parameter to be minimized in the algorithm. The smaller ρ is, the more precise results are returned. Therefore, we choose a radius r such that ρ is minimized, formally,

$$\hat{r} = \arg\min_{r} \frac{\ln 1/p'_1}{\ln 1/p'_2}. \tag{4.16}$$

SRS [129] proposed a tiny index to support approximate nearest neighbor queries in the ℓ_2 space. It is reported that the index size of SRS is at least one order of magnitude smaller than that of C2LSH [55]. SRS projects data points from the original high-dimensional space into a low-dimensional space via 2-stable projections. The major observation is that the square of the ratio of the ℓ_2 distance between two points in the projected space to the ℓ_2 distance between them in the original space follows the standard chi-squared distribution. Therefore, SRS can index the data points in the low-dimensional projected space and process approximate nearest neighbor queries for high-dimensional points with theoretical guarantees based on the chi-squared distribution. However, such 2-stable projection restricts SRS to build its

index in the ℓ_2 space. In contrast, our LazyLSH method is proposed based on the intuition that given $p > 0$ and $s > 0$, if two points are close in the ℓ_p space, then they are likely to be close in the ℓ_s space as well. Let $\epsilon = |p - s|$. The property holds with a higher probability for a smaller ϵ. In order to better support the approximate queries in fractional spaces, we materialize the base index in the ℓ_1 space.

To test whether the index built in the ℓ_2 space can answer approximate queries in fractional spaces, we try to use an ℓ_2 ball to approximate an $\ell_{0.5}$ ball. We set the approximate ratio $c = 3$. Our experimental result showed $p'_1 < p'_2$ when the dimensionality is greater than five, which means that the LSH functions in the ℓ_2 space might not necessarily be locality-sensitive in the $\ell_{0.5}$ space when the dimensionality increases. Based on the analysis above, we conclude that it is hard to integrate SRS into LazyLSH to support queries in fractional distances because SRS has its restriction on building its index in the ℓ_2 space. However, if there is a subsequent method which can build a tiny index in the ℓ_1 space, it should be easy to extend LazyLSH to the method.

4.4 Experiments

In this section, we study the performance of LazyLSH with various datasets. We mainly focus on the following two issues: (1) How the index size changes with different parameter settings. (2) How LazyLSH performs with respect to the efficiency and effectiveness on various datasets.

4.4.1 Datasets and Queries

In the experiments, synthetic datasets and four real datasets are used: **Mnist**[2] [84], **Inria**[3] [72], **LabelMe**[4] [119] and **Sun**[5] [138]. The statistics of the synthetic datasets and real datasets are summarized in Tables 4.3 and 4.4 respectively.

Synthetic datasets: We use synthetic datasets to study the influences of the dimensionality and the cardinality. We first fix the cardinality to be 40,000 and generate datasets with different dimensionality as {100, 200, 400, 800, 1600}. Then we fix the dimensionality to be 400 and generate datasets with different cardinality as {100k, 200k, 400k, 800k, 1.6m}. The value of each dimension is an integer randomly chosen from [0, 10000]. For each synthetic dataset, we randomly generate 50 query points as a query set.

Inria: The Inria dataset contains 1,491 holiday photos. 4,455,091 SIFT features [92] are extracted from the images, and each feature is represented as a 128-dimensional point. The value of each dimension is an integer in the range of [0, 255]. We randomly select 50 feature points as our query set and remove those features from the dataset during the query processing to avoid returning the same feature.

SUN: SUN is a dataset containing 108,753 images, each of which is attached with a class label to indicate which scene category the image belongs to. We obtain the GIST feature [47] of each image and generate a corresponding 512-dimensional data point. The value of each dimension is normalized to be an integer in the range of [0, 10,000]. We randomly pick 50 data points as a query set.

LabelMe: LabelMe is a dataset containing 207,909 images. We apply the processing method of SUN to the LabelMe dataset. Hence, the size of LabelMe is

[2] http://yann.lecun.com/exdb/mnist/

[3] http://lear.inrialpes.fr/∼jegou/data.php

[4] http://labelme.csail.mit.edu/Release3.0/

[5] http://sundatabase.mit.edu/

207,859.

Mnist: Mnist is a dataset consisting of 60,000 pictures of handwritten digits. Each picture has a class label representing which digit the picture shows. Each picture is represented as a 784-dimensional point, of which each dimension is an integer ranging from 0 to 255. In addition, Mnist contains a test set of 10,000 points. We randomly choose 50 points to form a query set.

Table 4.3: Parameter Settings for the synthetic datasets

Notation	Description	Values
$\lvert\mathcal{D}\rvert$	Cardinality	100k, 200k, <u>400k</u>, 800k, 1.6m
d	Dimensionality	100, 200, <u>400</u>, 800, 1600
c	Approximate ratio	2, <u>3</u>, 4, 5, 6
p	Supported ℓ_p space	<u>0.5</u>, 0.6, 0.7, 0.8, 0.9, 1.0

Table 4.4: Statistics of the real datasets and index sizes

Dataset	d	♯ points	value range	$\eta_{0.5}$	index size(MB)
Inria	128	4,455,041	[0, 255]	1358	23824
SUN	512	108,703	[0, 10,000]	916	1100
LabelMe	512	207,859	[0, 10,000]	959	2061
Mnist	784	60,000	[0, 255]	845	498

4.4.2 Evaluation Metrics

We follow the previous methods [132, 55, 56, 129] and adopt three metrics in our evaluations.

Space Consumption. The space is measured by the number of required hash tables and the index size.

Query Efficiency. The query efficiency is measured by the average number of I/Os of answering a query. If a block of an inverted list (4KB per block) is loaded into memory, the number of simulated I/Os (sequential) is increased by 1. If an

object is visited to compute its distance to the query, the number of simulated I/Os (random) is increased by 1.

Overall Ratio. The approximate ratio is defined as how many times farther a reported neighbor is compared to the real nearest neighbor. Formally, for a $\mathcal{N}_p(\vec{q}, k, c)$ query, $\{\vec{o_1}, \ldots, \vec{o_k}\}$ are the reported results sorted in ascending order of their distances to $\vec{q}$. Let $\{\vec{o_1^*}, \ldots, \vec{o_k^*}\}$ be the true kNNs sorted in ascending order of their distances to $\vec{q}$. The approximate ratio is calculated as:

$$\frac{1}{k} \sum_{i=1}^{k} \frac{\ell_p(\vec{o_i}, \vec{q})}{\ell_p(\vec{o_i^*}, \vec{q})} \tag{4.17}$$

The I/O cost and the overall ratio are averaged over queries. Unless otherwise specified, we materialize $\eta_{0.5}$ hash functions for the index so that queries can be answered in the ℓ_p spaces, where $0.5 \leqslant p \leqslant 1$. By default, queries are issued in the $\ell_{0.5}$ space.

Competitors. To the best of our knowledge, LazyLSH is the first work of supporting approximate nearest neighbor queries on multiple distance functions with a single index. Moreover, existing approaches do not support fractional distance metrics. There is no direct competitor of our approach. Alternatively, we modify C2LSH [55] and SRS [129] as competitors.

C2LSH: We build the index of C2LSH in the ℓ_1 space. Then we retrieve $(k + 100)$ candidates in the ℓ_1 space, and select the top-k points from the candidate set with the smallest ℓ_p distance to the query.

SRS: We build the index of SRS in the ℓ_2 space because SRS uses the 2-stable distribution. We retrieve the candidates in the ℓ_2 space, and select the top-k points from the candidate set with the smallest ℓ_p distance to the query. The number of projected dimensions in SRS is set to 6 as the experimental setting in [129]. The approximate ratio c is set to 3 for comparison.

Implementation. Our algorithms were implemented in C++. All experiments were conducted on a PC with Intel Core i7-3770 CPU @ 3.40GHz, 8GB memory, 500GB hard disk, running Ubuntu 12.04LTS. The page size was set to 4 KB in the experiments.

4.4.3 Study on Synthetic Datasets

We use synthetic datasets to study the index size w.r.t. different parameter settings. As discussed in Section 4.2.3, the index size is affected by four parameters: (1) the cardinality of the dataset $|\mathcal{D}|$, (2) the dimensionality d, (3) the approximate ratio c, and (4) the range of supported ℓ_p spaces, where η_p hash functions are built for the base index. Table 4.3 shows the parameter settings of the synthetic datasets. The default values are underlined in the third column. We study the required index size w.r.t each parameter by varying one parameter and setting the other three parameters to the default values, as presented in Table 4.5.

Effect of the cardinality $|\mathcal{D}|$: Table 4.5a shows the index size w.r.t $|\mathcal{D}|$. When $|\mathcal{D}|$ increases, the number of required hash functions increases, and so does the index size.

Effect of the dimensionality d: Table 4.5b shows the index size w.r.t d. It is worth noting that the index size decreases when d increases. This is because η_p changes with $(\hat{p}'_1 - \hat{p}'_2)$ as shown in Equation 4.12 and $(\hat{p}'_1 - \hat{p}'_2)$ varies w.r.t different numbers of dimensions. Figure 4.8 shows the relationship between the value of $(\hat{p}'_1 - \hat{p}'_2)$ and the dimensionality. The solid line in this figure plots $(\hat{p}'_1 - \hat{p}'_2)$ when the approximate ratio $c = 3$. As can be seen, $(\hat{p}'_1 - \hat{p}'_2)$ first decreases rapidly with the dimensionality and reaches the smallest value when $d = 16$. Then $(\hat{p}'_1 - \hat{p}'_2)$ increases slowly when the dimensionality increases. This explains the reason why the index size decreases with d when $d > 100$ for the synthetic datasets. Please note that the horizontal axis in Figure 4.8 is shown on a logarithmic scale with base 2.

Table 4.5: Index size w.r.t. different parameter settings

(a) Index size vs. the cardinality $|\mathcal{D}|$

$\lvert\mathcal{D}\rvert$	100k	200k	400k	800k	1.6m
$\eta_{0.5}$	923	979	1025	1071	1116
Size(MB)	1063	2211	4557	9250	18291

(b) Index size vs. the dimensionality d

d	100	200	400	800	1600
$\eta_{0.5}$	1223	1108	1025	966	879
Size(MB)	5011	4778	4557	4360	3997

(c) Index size vs. the approximate ratio c

c	2	3	4	5	6
$\eta_{0.5}$	7114	1025	570	425	355
Size(MB)	31609	4557	2531	1889	1577
I/Os	672184	77532	37252	24922	18712
Ratio	1.011	1.053	1.075	1.084	1.089

(d) Index size vs. the range of supported ℓ_p spaces

p	0.5	0.6	0.7	0.8	0.9	1.0
η_p	1025	711	579	507	462	432
Size(MB)	4557	3157	2576	2252	2051	1916

Effect of the approximate ratio c: Table 4.5c shows the index size w.r.t c. When c increases, the index size decreases. The reason can be also explained by Figure 4.8. As shown in the figure, for a fixed dimension, $(\hat{p}'_1 - \hat{p}'_2)$ increases when c increases, which leads to the smaller index size. When $c = 2$, the number of required hash functions $\eta_{0.5}$ is around seven times of $\eta_{0.5}$ for $c = 3$.

In addition, we test the number of I/Os and the overall ratio when processing approximate queries with different approximate ratio c. We notice that: (1) the number of I/Os decreases significantly as c increases. The number of I/Os when $c = 2$ is about nine times of that when $c = 3$, because of the difference in index

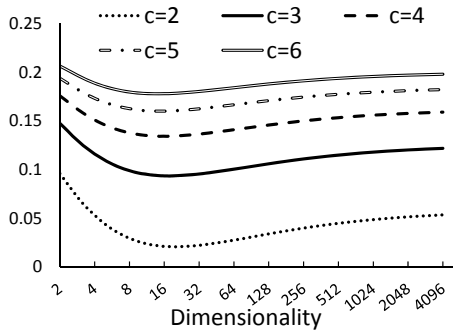


Figure 4.8: $(\hat{p}'_1 - \hat{p}'_2)$ w.r.t d in the $\ell_{0.5}$ space

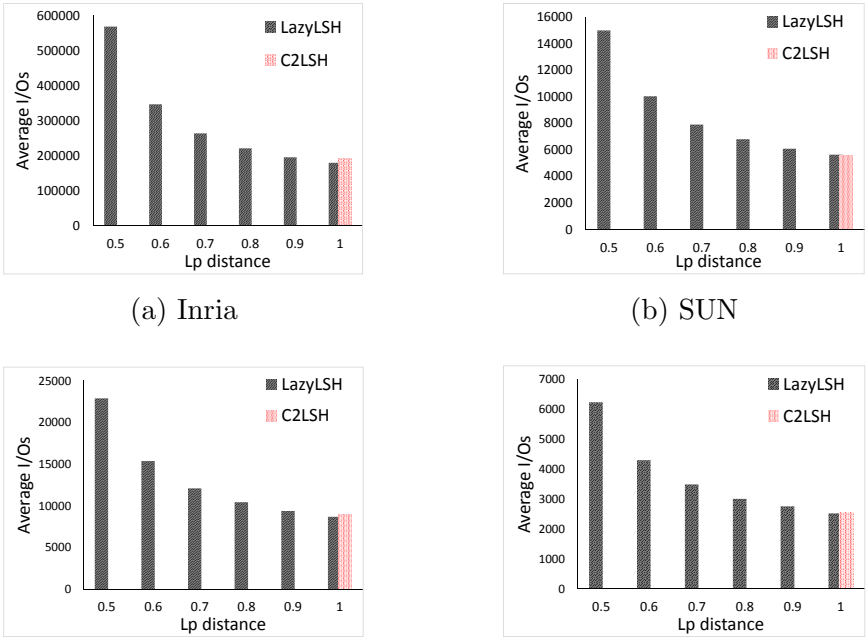
size. (2) the overall ratio increases with c because a larger approximation ratio is applied, which means that we can get more accurate results when we use a smaller c. Therefore, c can be viewed as a parameter to be set as a trade-off between query accuracy and efficiency (index size). To make it comparable to C2LSH, which used $c = 2$ or 3 in its experiment [55], we set $c = 3$ for LazyLSH in the rest of the experiments.

Effect of the range of supported ℓ_p spaces: Table 4.5d shows the index size w.r.t the range of supported ℓ_p spaces. To support a larger range of ℓ_p spaces, we need to materialize more hash functions. For instance, we need 2.37x hash functions to support queries in a range of ℓ_p spaces, where $0.5 \leqslant p \leqslant 1$, compared to the number of hash functions required for the single ℓ_1 space.

4.4.4 Study on Real Datasets

4.4.4.1 Index Size

We first study the index size for the real datasets. We materialize $\eta_{0.5}$ hash functions as the base index so that queries $\mathcal{R}_p(\vec{q}, \delta, c)$ can be supported in a range of ℓ_p spaces, where $0.5 \leqslant p \leqslant 1$. Table 4.4 shows the number of hash functions required and the index size for the real datasets. When the dimensionality increases, fewer hash functions are required. This observation is consistent with the result of the synthetic datasets shown in Table 4.5b.



(a) Inria (b) SUN

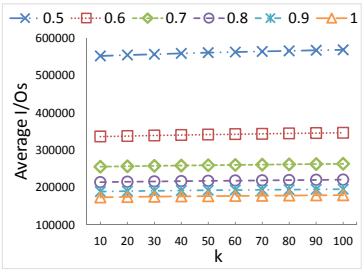
(c) LabelMe (d) Mnist

Figure 4.9: I/O costs on real datasets w.r.t. the ℓ_p distance

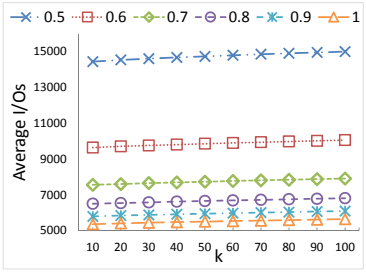
4.4.4.2 I/O Cost of Processing Queries

Next we study the performance of LazyLSH in terms of the I/O cost of processing a query.

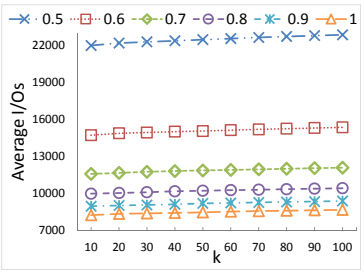
I/O Cost w.r.t. the query ℓ_p space: Figure 4.9 plots the average I/Os of processing a query w.r.t. the ℓ_p space where the number of nearest neighbors k is set to 100. The query processing in the $\ell_{0.5}$ space incurs more I/O overhead than the ℓ_1 space. Generally, a smaller p will lead to a higher I/O cost. This is because the processing of queries in the $\ell_{0.5}$ space requires a higher collision threshold for an object to become a candidate and more index entries are needed to read. In the ℓ_1 space, the performance of LazyLSH is similar to C2LSH. The average I/O costs for these two methods are at the same level. However, C2LSH can only support queries in the ℓ_1 space, while in contrast, LazyLSH is designed to support queries in a larger range of ℓ_p distances. Please note that the I/O cost of SRS is not reported as the



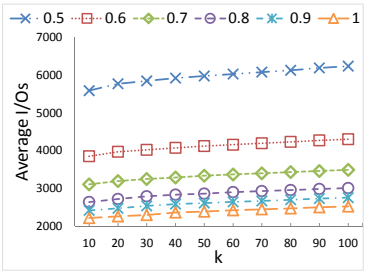
(a) Inria



(b) SUN



(c) LabelMe



(d) Mnist

Figure 4.10: I/O costs on real datasets w.r.t. the number of k

provided implementation[6] is in-memory and SRS is based on the ℓ_2 distance. Still, it is obvious that SRS can achieve the best performance in terms of I/O cost as its index size is one order of magnitude smaller than that of C2LSH as reported in [129]. Even though SRS has small I/O cost, its reliance on the 2-stable distribution in the ℓ_2 space constrains us from using its technique as the base index structure, which is explained in Section 4.3.4

I/O Cost w.r.t. the number of k: Figure 4.10 plots the average I/Os of processing a query w.r.t. the number of returned nearest neighbors. The horizontal axis k represents how many nearest neighbors are returned, ranging from 10 to 100. We observe that there is a slight increase of I/Os on all the four datasets when k increases. This indicates that users can issue a query with a larger k to get more precise nearest neighbors with a few additional I/Os.

[6] https://github.com/DBWangGroupUNSW/SRS

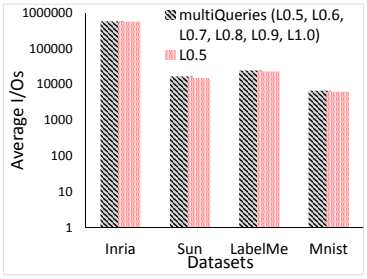


Figure 4.11: Multiple-query optimization

I/O Cost w.r.t. the multiple-query optimization:

One application of using LazyLSH is to select the optimal ℓ_p metric, which is shown to be highly application-dependent [54]. One way is to use classifier methods to select the best ℓ_p metric. In particular, we retrieve the approximate kNNs in different ℓ_p spaces. Then we select the optimal ℓ_p metric with the highest classification accuracy as shown in Table 4.1. This approach requires performing approximate kNN queries in different ℓ_p spaces, which can be processed as the multi-query optimization in Section 4.3.3.

Figure 4.11 presents the average I/Os of processing a single query versus the average I/Os of processing multiple queries concurrently. We try to find the kNNs of the same data point in different ℓ_p spaces (six queries in the $\ell_{0.5}$, $\ell_{0.6}$, $\ell_{0.7}$, $\ell_{0.8}$, $\ell_{0.9}$ and ℓ_1 spaces in this experiment). The queries of different ℓ_p spaces are processed together by sharing their I/Os, as they all probe some common hash buckets. As indicated in the figure, processing multiple queries concurrently only incurs a few more I/Os than processing a single query. This observation motivates us to process queries of different ℓ_p distances in a batch, instead of processing them individually. With the help of the multiple-query optimization, we can easily get the approximate kNNs of different ℓ_p metrics and choose the optimal ℓ_p metric for a particular dataset based on classification methods such as the kNN classifier as presented in Table 4.1 in the introduction.

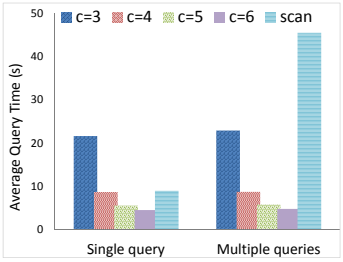


Figure 4.12: Time with multi-query optimization

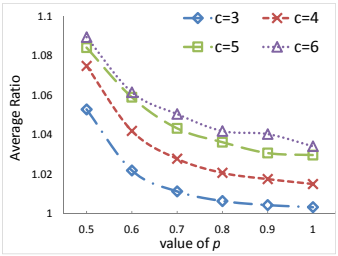


Figure 4.13: Average overall ratios w.r.t. c

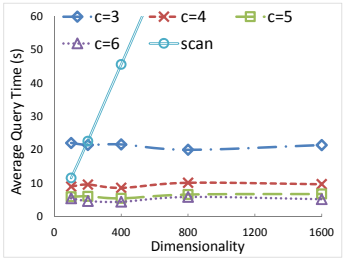


Figure 4.14: Query time w.r.t. dimensionality

4.4.4.3 Running Time of Processing Queries

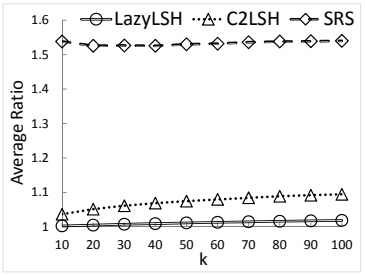
This experiment analyzes the query time of LazyLSH versus that of the linear scan method. As presented in Table 4.1 and Section 4.4.4.2, we need to retrieve approximate kNNs in different ℓ_p spaces to test the accuracy of the kNN classifier in order to pick the optimal ℓ_p metric.

We first study the effect of the multi-query optimization in terms of query time. We use the synthetic dataset with cardinality of 400k and dimensionality of 400 in this experiment. We set the approximate ratio $c = 3, 4, 5, 6$ and the number of returned points $k = 100$. The result for $c = 2$ is skipped as the I/O cost for $c = 2$ is much higher than others as shown in Table 4.5c. The single query is performed in the $\ell_{0.5}$ space, and the multiple queries are preformed in the $\ell_{0.5}$, $\ell_{0.6}$, $\ell_{0.7}$, $\ell_{0.8}$, $\ell_{0.9}$ and ℓ_1 spaces using our multi-query optimization. Figure 4.12 shows that the average query time of the two methods. The query time of LazyLSH with $c = 4$ for the single query is at the same level as that of linear scan. However, the running time of linear scan increases dramatically when multiple queries are answered. In contrast, the running time of LazyLSH for processing multiple queries remains at the same level as that for performing the single query. This finding is consistent to the result in Figure 4.11, because the number of I/Os increases by a small amount when processing the multiple queries.

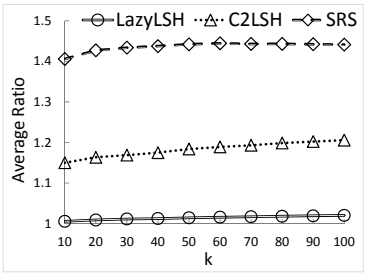
In addition, Figure 4.12 shows that the running time decreases as the approximate

ratio c increases. In fact, the choice of c can be viewed as a trade-off between accuracy and query time. As it is shown in Figure 4.15, LazyLSH achieves the highest accuracy among the LSH competitors in performing approximate queries using fractional distance metrics. We are interested in how the accuracy changes given different settings of c. Figure 4.13 presents the average overall accuracy ratio of performing queries w.r.t c in different ℓ_p spaces. The accuracy shown in the figure is fairly high. Even for $c = 6$, the overall ratio in the $\ell_{0.5}$ space is smaller than 1.1. This finding indicates that we can set c to be larger for faster speed while the accuracy remains acceptable. We find that $c = 5$ or 6 might be good choices considering the trade-off between accuracy and query time.

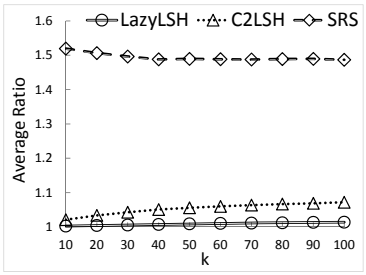
Next we study the query time with respect to the dimensionality. We use the synthetic datasets with cardinality of 400k and dimensionality of 100, 200, 400, 800 and 1600. Figure 4.14 presents the average running time for processing multiple queries with different dimensionality. The average running time of the linear scan method increases linearly with the dimensionality because the CPU time of computing the distance to the query grows linearly. Please note that the results of linear scan for $d = 800$ and 1600 are not shown for better viewing of the results of LazyLSH, and the results follow the trend. In contrast, the query time of LazyLSH remains at the same level for different dimensionality. When the approximate ratio is greater than or equal to 4, LazyLSH achieves better results in terms of query time compared to the linear scan method. In addition, LazyLSH gains more speedup when the number of dimensions increases, because the number of required hash functions does not increase linearly as presented in Table 4.5b. Again, the running time decreases given a larger approximate ratio c. In order to balance the accuracy and query time, we can use a larger c for those datasets with low dimensionality, and use a smaller c for those datasets with high dimensionality.



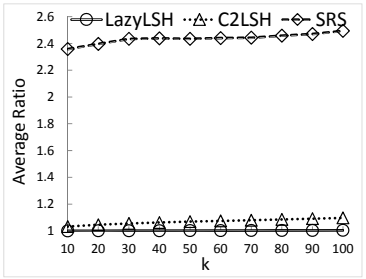
(a) Inria



(b) SUN



(c) LabelMe



(d) Mnist

Figure 4.15: Average overall ratios on real datasets

4.4.4.4 Overall Ratio

Overall Ratio for queries in fractional metrics: We then compare LazyLSH with existing work in the context of the overall ratio. Figure 4.15 presents the average overall ratio for queries in the $\ell_{0.5}$ space. In general, LazyLSH outperforms C2LSH in terms of the overall ratio. In most cases, the overall ratio of LazyLSH is less than 1.02. In contrast, C2LSH does not perform well for queries in the $\ell_{0.5}$ space, because C2LSH is designed to answer queries in the ℓ_1 or ℓ_2 spaces and it is not optimized to answer queries for fractional distances such as the $\ell_{0.5}$ distance. Therefore, LazyLSH has better performance in terms of the overall ratio.

Impact of the query-centric rehashing: Next we study the impact of the query-centric rehashing. To achieve a fair comparison, the queries are conducted in the same index with different rehashing methods. In particular, the queries are conducted in the ℓ_1 space and the approximate 100 NNs are returned. Figure 4.16

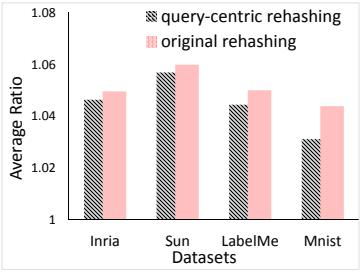


Figure 4.16: Impact of query-centric rehashing

plots the average overall ratio for different rehashing methods. We notice that our query-centric rehashing method outperforms the original rehashing method on all the four datasets. This is because the query-centric rehashing method is likely to retrieve nearby points, while the original rehashing method may retrieve distant points in some cases as stated in Figure 4.7.

4.5 Summary

In this chapter, we proposed an efficient mechanism called LazyLSH to answer approximate kNN queries under fractional distance metrics in high-dimensional spaces. We observed that an LSH function in a specific ℓ_p space can be extended to support queries in other spaces. Based on this observation, we materialized a base LSH index in the ℓ_1 space and used it to process approximate kNN queries in different ℓ_p spaces. We conducted extensive experiments on synthetic and real datasets, and the experimental results showed that LazyLSH provides accurate results and improves existing machine learning algorithms for retrieving approximate kNN under different fractional distance metrics.

Chapter 5

ARShop: An Interactive Shopping System using Augmented Reality

In this chapter, we introduce a system called ARShop, which is a one-stop solution to enhance the shopping experience for users by using the augmented reality (AR) technology. For mall or shop owners, ARShop provides them with cloud-based tools to digitize their shops. A shop owner can create a digital shop and upload images of the shop. The shop owner can further add annotations on their uploaded photos to indicate the shop's name and available products in the shop. For shoppers, ARShop provides them with a tool to input their shopping lists. Based on a shopping list, ARShop equips the INSPIRE framework described in Chapter 3 to perform spatial keyword queries to find the nearby shops, whose annotations match with the user-specific shopping list. In addition, ARShop provides shoppers with a mobile application to search for items in their shopping lists by taking photos of their surrounding environments. ARShop takes the user-uploaded photo as a query and finds the similar photos in the database using the LazyLSH method described in Chapter 4. Annotations of the user-interested items are popped up on the similar photos.

5.1 Motivation

We are living in the cyber-physical world, where the physical space and the virtual space interact simultaneously. While the physical space is virtually enhanced with information, the virtual space is continuously refreshed with real-world information [106]. For example, important events in the physical world are captured through mobile devices and materialized within the virtual world. Correspondingly, events within the virtual world, such as a sales promotion, can affect the physical world.

The development of Augmented Reality (AR) technologies has made it possible to link these two spaces together into a co-existing space. We are on the verge of ubiquitously adopting AR technologies to enhance our perception of reality in new and enriched ways. AR combines real and virtual objects in a real environment. It is a live view of the physical world whose elements are augmented by the computer-generated sensory input such as videos, graphics and GPS data.

Over the past few decades, researchers and developers have found many areas that could benefit from the augmentation [134]. With today's high-end smart phones and the latest AR technologies, such as Google Glass[1], Microsoft HoloLens[2] and Oculus Rift[3], artificial information about the environment can be overlaid on the real world. The information about the surrounding real world becomes interactive and digitally manipulable.

Among the AR applications, the AR technology used in landmark recognition has become an active research topic in the last decade [128, 150, 69]. Landmark recognition can determine where a photo is taken based on the image database collected from the Web. The integration of landmark recognition and the AR technology can be used in tourism and sightseeing to suit the convenience of tourists. In such AR

[1] http://www.google.com/glass/

[2] https://www.microsoft.com/microsoft-hololens/

[3] https://www.oculus.com/

applications, information of tourist attractions is presented on the screen of a user's mobile or tablet. For example, when a user visits a tourist attraction, he/she wants to search for the introductory information about the attraction. He/she takes a photo of the attraction. Then the introductory description is popped up and augmented on the photo.

Besides Landmark recognition, AR applications have been used in various fields such as gaming [109] and navigation [113, 103]. In this chapter, we narrow our scope of AR applications in the context of navigation. In particular, we focus on the shopping scenario in order to enhance the shopping experience by using the AR technology. Consider the following shopping scenario.

Example 5.1. *A user goes shopping in an unfamiliar shopping mall. He/she has a shopping list and wants to buy the items in the shopping list. In order to quickly locate those items, the user launchs an AR application and takes a photo of his/her surrounding environment. Based on the user-uploaded image, the AR application analyses the image and directs the user to find the items in the shopping list.*

As stated in the previous example, such an AR application can enhance users' shopping experience. **Our goal is to build an AR system, which can direct a user to find the items in the shopping list by recognizing the photo of the user's surroundings.** To this end, we present ARShop, which is a one-stop solution to enhance the shopping experience for users by using the AR technology. A user can either input a shopping list in a Web interface or a mobile application. When he/she enters a shop, he/she takes a photo of his/her surroundings. The application can infer the current location of the user and direct him/her to find the items in the shopping list.

To fulfill our goal, an ideal system should have the following features, which are also the motivations of the ARShop system.

Feature 1: Capturing photos with enriched location information. Location information can help users find a wide variety of location-specific information. In outdoor environments, we can easily retrieve the longitude and latitude using GPS. Due to the complexity of electromagnetic wave propagation in indoor environments, the GPS approaches used for outdoor positioning become less accurate when used indoor. Instead, other positioning methods, such as Wi-Fi positioning, are used in indoor environments [11]. Recent studies also show that with the help of compass which measures the Earth's magnetic field, indoor positioning system achieves even higher accuracy [36, 100].

Current mobile devices have built-in sensors that measure location, orientation, and various environmental conditions. We provide a mobile application to record the information of GPS, Wi-Fi and compass. When a user takes a photo using the mobile application, the location information is attached to the photo. For shop owners, they can use the mobile application to take photos in their shops. When they upload the images to the server, the location information is also attached. For shoppers, the location information can help to retrieve nearby images from the database.

Feature 2: Effective shop management. First of all, the ARShop system requires shop owners to digitize their shops. A effective shop management tool should be provided for shop owners to manage their shops.

A shop owner can create a digital shop and then upload images of the shop to the server using the mobile application as mentioned previously. The shop owner can also add annotations on the images to point out the shop's name and the available products in the images. These annotations are used to match with a user's shopping list. If an annotation matches with an item in the user's shopping list, it indicates that the shop probably has a product, which is of interest to the user. Furthermore, shop owners can also record linkages between different images to indicate that same objects exist in these images. This operation can help us to link the images of a shop

together to form a virtual model of the shop. In this way, shop owners can manage and digitalize their shop effectively.

Feature 3: Efficient query processing to achieve the real-time human-computer interaction. If there exists an annotation that matches with a user's shopping list, this annotation should be popped up in association with the image on the mobile screen in real time. Such operations must be processed efficiently to achieve the real-time human-computer interaction.

In the context of our AR application, there are two types of time-consuming operations.

1. Spatial keyword search: it is used to map the items in a user's shopping list to the images' annotations. As the images are geo-tagged, the annotations are also attached with a location to reduce the query cost.

2. Spatial image search: images are usually represented as high-dimensional points. Retrieving similar images of a query image is actually preforming the nearest neighbor search in high-dimensional spaces.

For the first type of operations, we use the INSPIRE framework described in Chapter 3 to speed up the processing of spatial keyword queries. For the second type of operations, we use the LazyLSH method described in Chapter 4 to speed up the processing of approximate nearest neighbor queries in high-dimensional spaces. Therefore, our system can handle these two types of operation efficiently and fulfill the real-time human-computer interaction requirement.

Integrating all the above features, we demonstrate ARShop as shown in Figure 5.1. The major advantages of ARShop are summarized as follows:

1. ARShop holds a website and a mobile application to interact with users. ARShop also provides user-friendly interfaces for users to manage and browse a virtual shop.

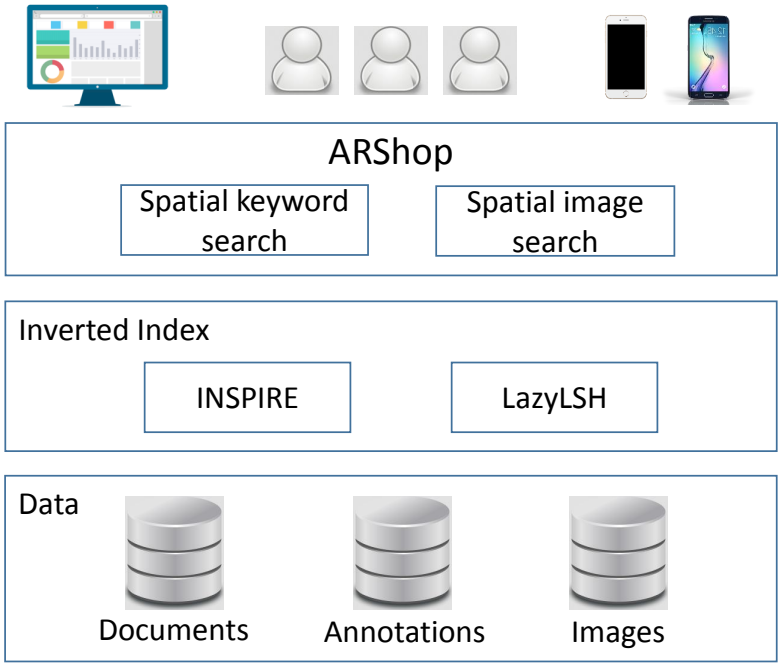


Figure 5.1: Overview of ARShop

2. ARShop provides a mobile application to direct users to the items in their shopping lists using the AR technology.

3. ARShop equips advanced indexing techniques, including INSPIRE and LazyLSH, to support different types of queries efficiently to achieve real-time human-computer interaction.

5.2 System Architecture

The ARShop system adopts the client-server architecture, which is shown in Figure 5.2. On the client side, we build a website and create an Android application for both shop owners and shoppers. On the server side, we build a data management tool that supports the storage and retrieval of data.

On the client side, ARShop has a website and a mobile application. Shop owners can use the website to manage their shops. They can upload images of their

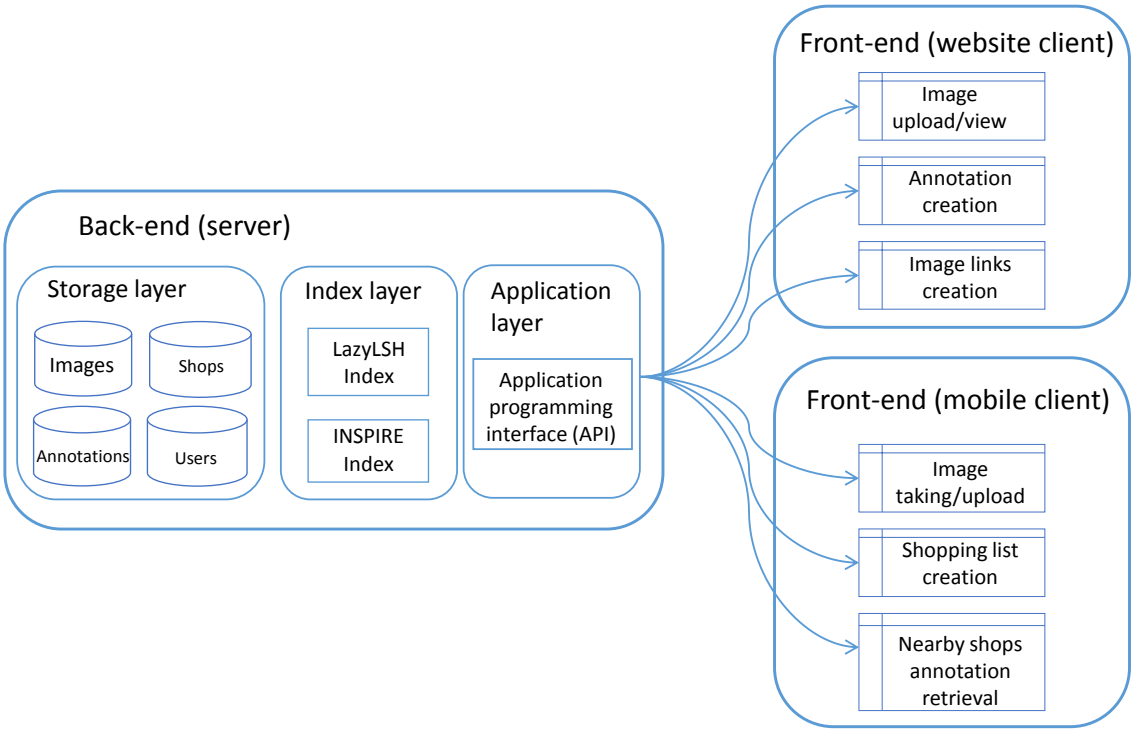


Figure 5.2: System architecture of ARShop

shops and add annotations on the images to specify available products in the shops. Furthermore, shop owners can also record linkages between different images to indicate that same objects exist in these images. For shoppers, they can use the mobile application to input their shopping lists and upload photos of their surroundings as queries. Then the mobile application displays the similar nearby images, whose annotations match with the items in users' shopping lists.

The server consists of three layers: (1) the storage layer, (2) the index layer and (3) the application layer.

- In the storage layer, data such as users, shops and annotations are stored in MySQL, while the image files are stored in the file system.
- In the index layer, inverted indexes are built to support the location-based services.

- In the application layer, application programming interfaces (APIs) are defined and implemented to enable the client-server interactions.

5.2.1 Storage Layer

In the storage layer, we use MySQL to store the information of users, shops, images and annotations.

1. Each shop is created by a shop owner, which is also a user of the system. The shop owner can upload images to the shop.

2. Each image is represented as a Bag-of-Word vector as well as a number of low-level SIFT features [91].

3. Each image has its GPS, orientation as well as WIFI signal information when it is taken.

4. Each annotation has its content stored as text. It is associated with an image and has its position in the image stored in the database.

Correspondingly, an ER diagram is shown in Figure 5.3.

5.2.2 Index Layer

In order to support fast query processing, inverted indexes are built in the index layer. ARShop makes use of the two types of inverted indexes proposed in this thesis to support location-based services.

Shop owners can add annotations onto images to indicate the available products. These annotations are stored as text and used to match with a user's shopping list. In order to achieve fast keyword search when retrieving relevant annotations that match with a user's shopping list, these annotations are indexed using the proposed INSPIRE method, which is introduced in Chapter 3.

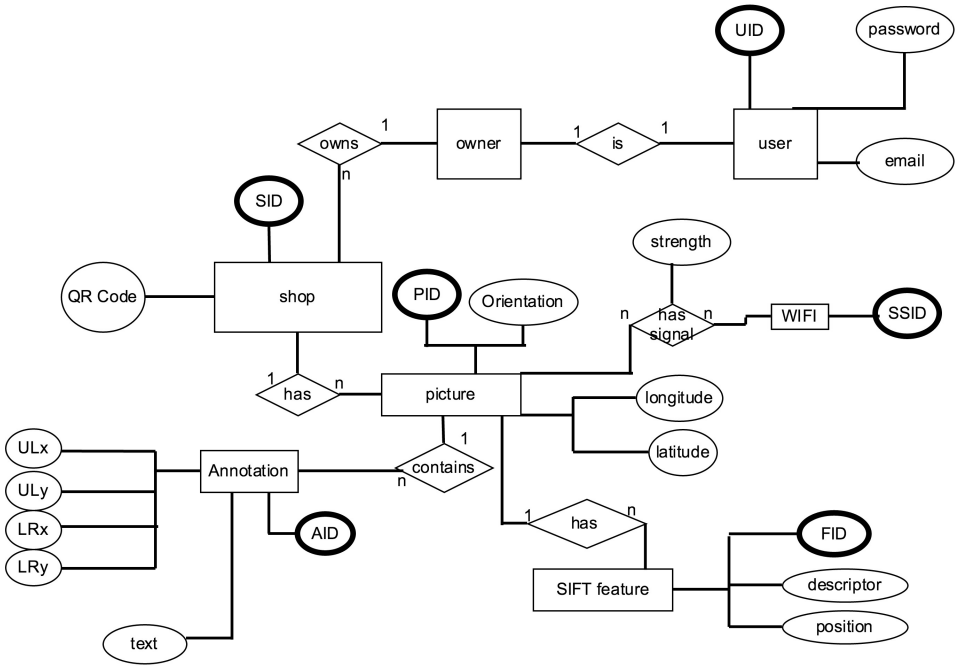


Figure 5.3: ER diagram of ARShop

When a user submits a query image, the similar images are retrieved. This processing is conducted by the LazyLSH method, which is introduced in Chapter 4. Correspondingly, the LazyLSH index is needed to build in order to achieve fast kNN processing.

5.2.3 Application Layer

The application layer provides APIs for (1) uploading/retrieving images, (2) uploading/retrieving annotations of an image and (3) most importantly retrieving similar images.

The "retrieving similar images" API requires the client to upload an image of a user's surroundings. The server first uses the location information associated with the uploaded image as well as the user's shopping list to filter out irrelevant images in the database. After that the server extracts the high-dimensional representation of the uploaded image, and finds the similar images in the database. The relevant

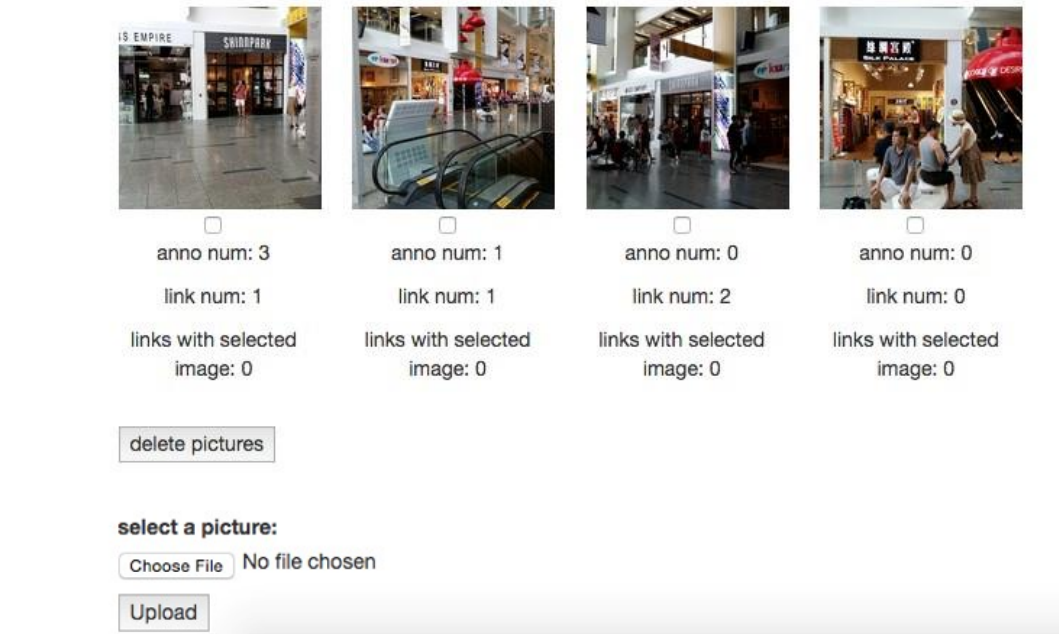


Figure 5.4: Overview of the Website

images are then returned in the descending order of similarities.

5.3 Demonstration

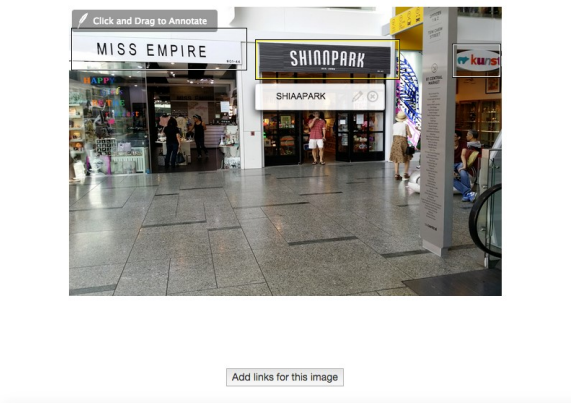
The ARShop system contains a Web interface and a mobile application. In this section, we demonstrate the ARShop system.

5.3.1 ARShop Website

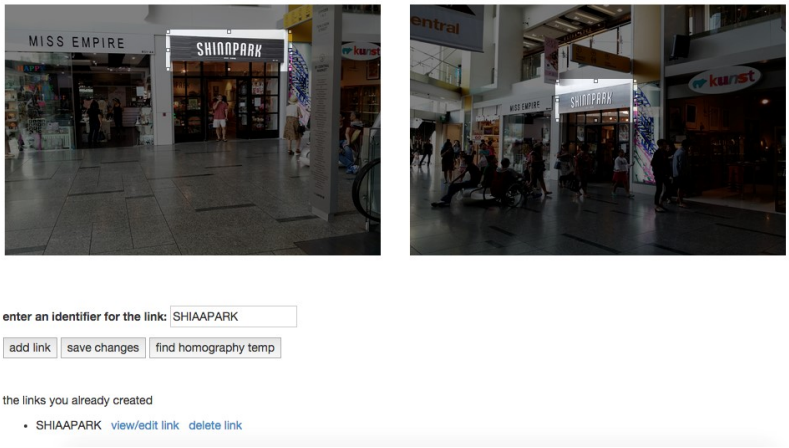
The ARShop website[4] is a Web interface for shop owners. It currently consists of three features, namely (1) uploading images, (2) adding annotations to images and (3) adding links between images.

Figure 5.4 presents an overview of the website. Shop owners can manage their shops using this website. For example, shop owners can create digital shops and upload images to shops. In addition, they can also delete images of a shop. This function is used when a shop is decorated and enables shop owners to upload a new

[4] http://www.shopbyar.com/ARShop/



(a) Adding Annotations



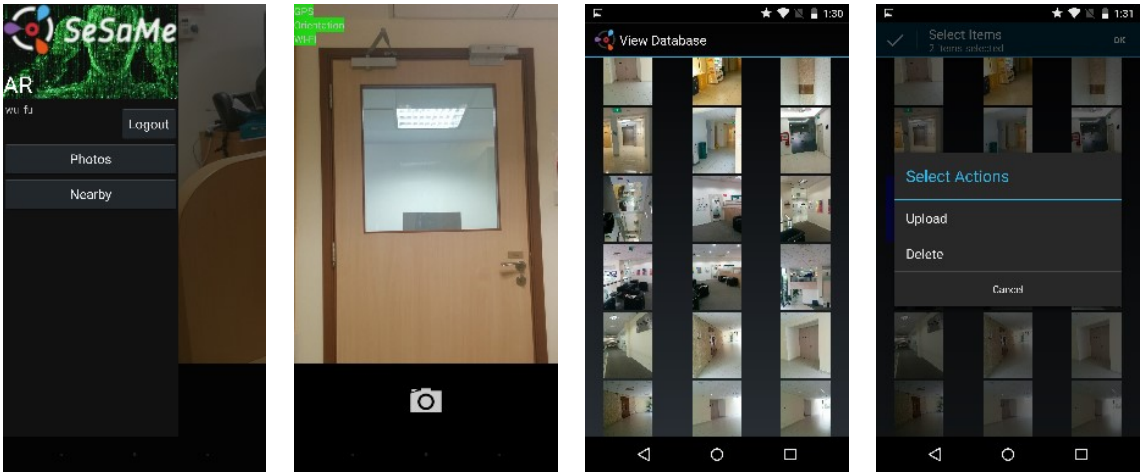
(b) Adding links

Figure 5.5: Web Interfaces

set of images of the shop.

When shop owners click one of the image thumbnails, the full image will be shown, and they can select some areas in the image and add annotations to them, as shown in Figure 5.5a. Besides the name of the shop, shop owners can enter detailed information about the items for sale. This function can make it easier for the annotations to match with the items in shoppers' shopping lists.

Shop owners can also add links between images by clicking the "add link" button in Figure 5.5a, and it will bring them to the "add link" mode shown in Figure 5.5b.



(a) Main Menu (b) Taking Image (c) Gallery (d) Upload Images

Figure 5.6: Mobile application for shop owners

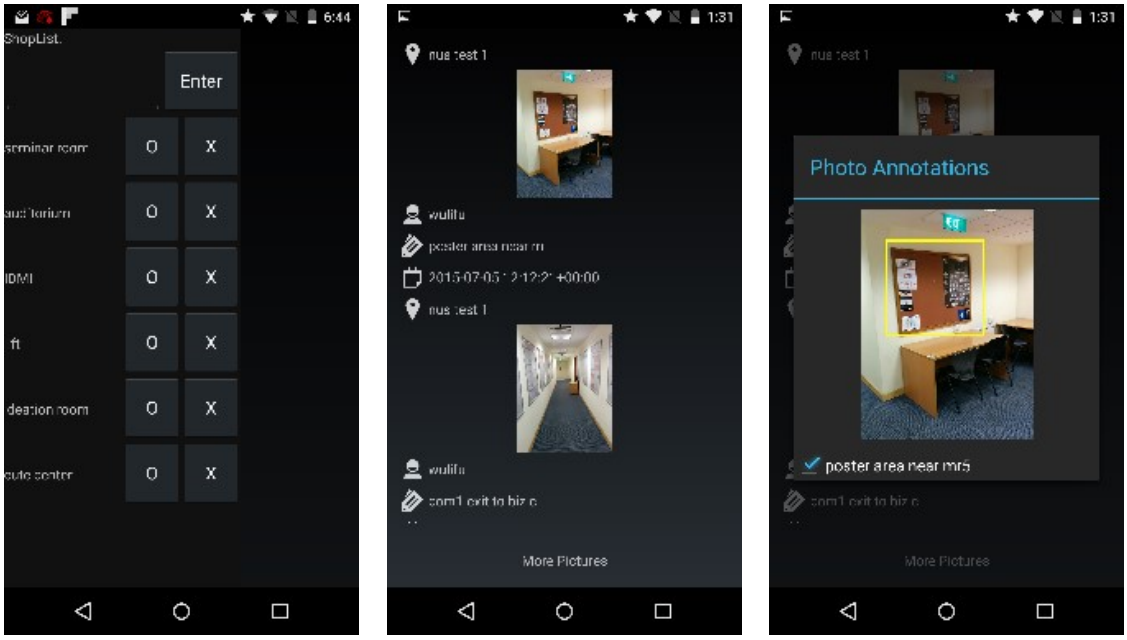
In the "add link" mode, shop owners can select overlapping areas between images and add them as "links". The links added by shop owners will help to perform geometric reasoning among images in the database.

5.3.2 Mobile Application

While the ARShop website is for the shop owners to digitize and manage their shops, our ARShop mobile application is the key to enhance the shopping experience of shoppers. Currently, it implements two functionalities for shoppers: (1) searching nearby images by a shopping list and (2) searching nearby images by taking a photo of surrounding environments. Additionally, this mobile application also allows shop owners to take and upload images.

For shop owners:

Figure 5.6a shows the main interface of the application. When swiping to the left, shop owners enter the interface for taking photos, as shown in Figure 5.6b. The photos taken will be associated with GPS, WIFI and orientation information and stored in the phone. Later the shop owners can view these photos by pressing the



(a) Shopping List (b) Search by Shopping List (c) Search by Image

Figure 5.7: Mobile application for shoppers

"Photos" button in the main interface, as shown in Figure 5.6c. Then they can select the images by long tapping them to enter the selection mode, upon completion they can choose to upload or delete the selected images, as shown in Figure 5.6d.

For shoppers:

When shoppers tap on the "Nearby" button in the main menu, they will be asked to input items into the shopping list, as shown in Figure 5.7a. After that the application will retrieve the nearby images, which have annotations matching with the items in the shopping list, as shown in Figure 5.7b. This query is conducted by the INSPIRE framework mentioned in Chapter 3. Besides searching nearby images in the shopping list, users can also take a picture of the surroundings, upload it to the server and the server will analyse the picture and return similar images with annotations, as shown in Figure 5.7c. This is performed by the LazyLSH method introduced in Chapter 4.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we studied advanced interactive methods using inverted index in the context of location-based services. Two types of location-based services are studied, including the location-based keyword search and the location-based image search. For the location-based keyword search, we combined autocompletion and spelling correction, which are two types of interactive search methods, to improve the search effectiveness. For the location-based image search as a new search method in interactive search, we studied the approximate search to quickly return similar results to improve the efficiency.

We surveyed the interactive search techniques used in location-based services in Chapter 2. In particular, we reviewed the existing work of query recommendation to facilitate interactive keyword search. Then, we focused on the interactive keyword search in the context of geo-textual databases including spatial keyword search and its various extensions. We also reviewed the existing methods of approximate nearest neighbor search, especially the locality-sensitive hashing method.

For the location-based keyword search, we presented our work on the interactive spatial keyword search in Chapter 3. We proposed INSPIRE, a general framework, which adopts a single, unifying strategy for processing different variants of spatial

keyword query. In particular, we observed that the relaxation can be done by expanding the spatial region or loosening the prefix matching constraint. For loosening string matching constraint, we adopted an incremental way to perform relaxation to maximize query reusability. The relaxation is applied by first allowing exact substring matching, then approximate prefix matching followed by approximate substring matching to maximize query reusability. Then we built a one-size-fits-all index to support all types of relaxation. To accelerate query processing, we proposed our intra-query optimization covering common result reuse and selectivity estimation. Moreover, we observed that the search-as-you-type paradigm allows appending characters after the initial query, so we proposed the inter-query optimization to reuse the results of the initial query in processing its appending query, instead of processing it from scratch. As a result, INSPIRE can decide the most appropriate relaxation for a spatial prefix query, and support fuzzy type-ahead search to provide realtime human-computer interaction. We demonstrated the efficiency, practicality and effectiveness of our proposed approaches using a comprehensive experimental evaluation.

For the location-based image search, we presented our work on supporting the approximate nearest neighbor search under fractional distance metrics in high-dimensional spaces in Chapter 4. We observed that an LSH function in a specific ℓ_p space could be extended to support queries in other spaces. Based on this observation, we proposed LazyLSH, which materializes a base LSH index in the ℓ_1 space and use the materialized index to process approximate kNN queries in different ℓ_p spaces. Extensive experiments on synthetic and real datasets show the efficiency and effectiveness of our proposed method which provides accurate results under fractional distance metrics. Using LazyLSH, existing machine learning algorithms can be improved for retrieving approximate kNN under different fractional distance metrics.

In Chapter 5, we demonstrated ARShop, which is an interactive augmented reality system for shopping. ARShop provides location-based services including spatial

keyword search and spatial image search. For shop owners, ARShop provides them with a Web interface to digitize their shops. For shoppers, they can input shopping lists and issue queries using photos of their surrounding environments using our mobile application. Based on the shopping list and the photo of the user's surrounding environment, the ARShop system can direct the user to shops that contain items in his/her shopping list.

6.2 Future Work

Although our system provides efficient solutions to support advanced interactive search in the context of location-based services, there still exist situations that it cannot handle well.

For the spatial keyword search, our INSPIRE method uses a fixed strategy to perform query relaxation. Users are required to specify some parameters before issuing queries, which might be difficult for the users without background knowledge. In the future work, we would like to investigate other relaxation strategies such as relaxing the edit distance threshold progressively so that similar results are returned first. Another possible improvement on spatial prefix search is to take semantics into account and organize results in appropriate orders by ranking and diversifying results. In addition, we can extend the query relaxation to other applications such as searching trajectories or graph data.

For the spatial image search, our LazyLSH method can support the approximate nearest neighbor search under fractional distance metrics in high-dimensional spaces. The ℓ_p metrics essentially aggregate the difference between each dimension of the two objects. This approach has two drawbacks:

1. It leaves many partial similarities uncovered since the distance computation is based on the fixed set of dimension.

2. The distance is often affected by a few dimensions with high dissimilarity.

In the future work, we aim to study partial similarities, which can also address the concentration problem in high-dimensional spaces. Next we plan to study how to support exact or approximate kNN search in a user-specific subspace or under a user-specific weighted distance function.

For the ARShop system, even though we can find similar images given a user-uploaded photo, the recognition accuracy is still a problem. There exist cases that two images are close in their high-dimensional representation, but they are in fact not similar when judged by humans. How to improve the recognition accuracy for image search is still an on-going research problem. In addition, we will use spatial information to help search similar images. We will investigate techniques like indoor positioning using Wi-Fi signals to infer the indoor position of a user. We plan to use a two-level clustering method to group images in the database, which first clusters images by Wi-Fi signals then further clusters images by their virtual features. Using such a method, we can quickly find similar images for a query image with its location information including GPS and Wi-Fi signals.

Bibliography

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD*, pages 13–24. ACM, 1999.

[2] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space. In *ICDT*, volume 1973, pages 420–434. Springer, 2001.

[3] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC*, pages 557–563. ACM, 2006.

[4] S. Alsubaiee, A. Behm, and C. Li. Supporting location-based approximate-keyword queries. In *SIGSPATIAL GIS*, pages 61–70. ACM, 2010.

[5] A. Andoni and P. Indyk. Efficient algorithms for substring near neighbor problem. In *SODA*, pages 1203–1212. Society for Industrial and Applied Mathematics, 2006.

[6] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *COMMUNICATIONS OF THE ACM*, 51(1):117–122, 2008.

[7] Y. Arens, C. A. Knoblock, and W.-M. Shen. *Query reformulation for dynamic information integration*. Springer, 1996.

[8] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280. Society for Industrial and Applied Mathematics, 1993.

[9] J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. C. Jain, and C.-F. Shu. Virage image search engine: an open framework for image management, 1996.

[10] M. Bader. *Space-Filling Curves: An Introduction With Applications in Scientific Computing*, volume 9. Springer, 2012.

[11] P. Bahl and V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM*, volume 2, pages 775–784. IEEE, 2000.

[12] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371. ACM, 2006.

[13] S. Basu Roy and K. Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *SIGMOD*, pages 361–372. ACM, 2011.

[14] H. Bay, T. Tuytelaars, and L. Gool. Surf: Speeded up robust features. In *ECCV*, pages 404–417. Springer, 2006.

[15] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[16] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *SIGKDD*, pages 407–416. ACM, 2000.

[17] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *ICDT*, pages 217–235. 1999.

[18] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, pages 39–48. ACM, 2003.

[19] K. Binder and D. Heermann. *Monte Carlo simulation in statistical physics: an introduction*. Springer Science & Business Media, 2010.

[20] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[21] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[22] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW*, pages 1157–1166. ACM, 1997.

[23] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, and R. Kimmel. Partial similarity of objects, or how to compare a centaur to a horse. *International Journal of Computer Vision*, 84(2):163–183, 2008.

[24] G. Calafiore, F. Dabbene, and R. Tempo. Uniform sample generation in lp balls for probabilistic robustness analysis. In *CDC*, volume 3, pages 3335–3340. IEEE, 1998.

[25] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *SIGKDD*, pages 875–883. ACM, 2008.

[26] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.

[27] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336. ACM, 1998.

[28] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388. ACM, 2002.

[29] S. Chaudhuri and G. Das. Automated ranking of database query results. In *CIDR*. Citeseer, 2003.

[30] S. Chaudhuri, V. Ganti, and L. Gravano. Selectivity estimation for string predicates: overcoming the underestimation problem. In *ICDE*, pages 227–238. IEEE, 2004.

[31] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD*, pages 707–718. ACM, 2009.

[32] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod. Inverted index compression for scalable image matching. In *Data Compression Conference*, pages 525–525, 2010.

[33] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *VLDB*, 6(3):217–228, 2013.

[34] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: Towards verification-free query processing on graph databases. In *SIGMOD*, pages 857–872. ACM, 2007.

[35] S.-L. Chuang and L.-F. Chien. Towards automatic generation of query taxonomy: a hierarchical query clustering approach. In *ICDM*, pages 75–82. IEEE, 2002.

[36] J. Chung, M. Donahoe, C. Schmandt, I.-J. Kim, P. Razavai, and M. Wiseman. Indoor location sensing using geo-magnetism. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 141–154. ACM, 2011.

[37] J. G. Cleary. Analysis of an algorithm for finding nearest neighbors in euclidean space. *ACM Trans. Math. Softw.*, 5(2):183–192, 1979.

[38] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endow.*, 2(1):337–348, 2009.

[39] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, pages 605–614. IEEE, 2002.

[40] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Trans. on*, 13(1):21–27, 1967.

[41] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246. ACM, 2007.

[42] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262. ACM, 2004.

[43] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665. IEEE, 2008.

[44] R. De La Briandais. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), pages 295–298, 1959.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255, 2009.

[46] S. Dhar and U. Varshney. Challenges and business models for mobile location-based services and advertising. *Commun. ACM*, 54(5):121–128, 2011.

[47] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of gist descriptors for web-scale image search. In *CIVR*, pages 19:1–19:8. ACM, 2009.

[48] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Rec.*, 39(1):41–47, 2010.

[49] E. N. Efthimiadis. Query expansion. *Annual review of information science and technology*, 31:121–187, 1996.

[50] R. Finkel and J. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[51] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, 1995.

[52] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, 1995.

[53] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *CIKM*, pages 696–703. ACM, 2005.

[54] D. Francois, V. Wertz, and M. Verleysen. The concentration of fractional distances. *TKDE*, 19(7):873–886, 2007.

[55] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552. ACM, 2012.

[56] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi. Dsh: Data sensitive hashing for high-dimensional k-nnsearch. In *SIGMOD*, pages 1127–1138. ACM, 2014.

[57] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. pages 436–445, 1997.

[58] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390. ACM, 2009.

[59] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava. Using q-grams in a dbms for approximate string processing. *IEEE Data Eng. Bull.*, 24(4):28–34, 2001.

[60] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500. Morgan Kaufmann Publishers Inc., 2001.

[61] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW*, pages 11–20. ACM, 2009.

[62] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, pages 124–131. ACM, 2009.

[63] J. Guo, X. Cheng, G. Xu, and H. Shen. A structured approach to query recommendation with social annotation data. In *CIKM*, pages 619–628. ACM, 2010.

[64] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[65] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSDBM*, pages 16–25. IEEE, 2007.

[66] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB*, pages 506–515. VLDB Endowment, 2000.

[67] P. Howarth and S. Rüger. Fractional distance measures for content-based image retrieval. In *ECIR*, pages 447–456. Springer, 2005.

[68] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613. ACM, 1998.

[69] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof. From structure-from-motion point clouds to fast location recognition. In *CVPR*, pages 2599–2606. IEEE, 2009.

[70] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal*, 9(3):214–230, 2000.

[71] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[72] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, pages 304–317. Springer, 2008.

[73] H. Jgou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, 2010.

[74] S. Ji and C. Li. Location-based instant search. In *SSDBM*, pages 17–36. Springer, 2011.

[75] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*, pages 154–161. ACM, 2005.

[76] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396. ACM, 2006.

[77] Y. Kim and K. Shim. Efficient top-k algorithms for approximate substring matching. In *SIGMOD*, pages 385–396. ACM, 2013.

[78] P. Knees, T. Pohle, M. Schedl, and G. Widmer. A music search engine built upon audio-based and web-based similarity measures. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 447–454. ACM, 2007.

[79] R. K. V. Kothuri, S. Ravada, and D. Abugov. Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In *SIGMOD*, pages 546–557. ACM, 2002.

[80] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137. IEEE, 2009.

[81] J. Lamping and S. Baker. Determining query term synonyms within query context, Dec. 22 2009. US Patent 7,636,714.

[82] L. J. Latecki, R. Lakaemper, and D. Wolter. Optimal partial shape similarity. *Image and Vision Computing*, 23(2):227 – 236, 2005. Discrete Geometry for Computer Imagery.

[83] J. F. Lawless. Inference in the generalized gamma and log gamma distributions. *Technometrics*, 22(3):409–419, 1980.

[84] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[85] H. Lee, R. T. Ng, and K. Shim. Approximate substring selectivity estimation. In *EDBT*, pages 827–838. ACM, 2009.

[86] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.

[87] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266. IEEE, 2008.

[88] G. Li, S. Ji, C. Li, and J. Feng. Efficient fuzzy full-text type-ahead search. *The VLDB Journal*, 20(4):617–640, 2011.

[89] R. Li, B. Kao, B. Bi, R. Cheng, and E. Lo. Dqr: a probabilistic approach to diversified query recommendation. In *CIKM*, pages 16–25. ACM, 2012.

[90] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen. Sk-lsh: An efficient index structure for approximate nearest neighbor search. *PVLDB.*, 7(9):745–756, 2014.

[91] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157. Ieee, 1999.

[92] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[93] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961. VLDB Endowment, 2007.

[94] H. Ma, M. R. Lyu, and I. King. Diversifying query suggestion results. In *AAAI*, volume 10, 2010.

[95] B. Manjunath and W. Ma. Texture features for browsing and retrieval of image data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(8):837–842, 1996.

[96] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[97] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM*, pages 469–478. ACM, 2008.

[98] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2, 2009.

[99] E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4-5):345–374, 1994.

[100] P. Nagpal and R. Rashidzadeh. Indoor positioning using magnetic compass and accelerometer of smartphones. In *MoWNeT*, pages 140–145. IEEE, 2013.

[101] A. Nanopoulos, D. Rafailidis, M. M. Ruxanda, and Y. Manolopoulos. Music search engines: Specifications and challenges. *Information Processing and Management*, 45(3):392 – 396, 2009.

[102] M. Naphade, J. Smith, J. Tesic, S.-F. Chang, W. Hsu, L. Kennedy, A. Hauptmann, and J. Curtis. Large-scale concept ontology for multimedia. *MultiMedia, IEEE*, 13(3):86–91, 2006.

[103] W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, R. Mller, J. Wieghardt, H. H?rtner, and C. Lindinger. Augmented reality navigation systems. *Universal Access in the Information Society*, 4(3):177–187, 2006.

[104] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[105] G. Navarro and R. A. Baeza-Yates. A practical q -gram index for text retrieval allowing errors. *CLEI Electron. J.*, 1(2):273–282, 1998.

[106] B. C. Ooi, K. L. Tan, and A. Tung. Sense the physical, walkthrough the virtual, manage the co (existing) spaces: A database perspective. *SIGMOD Rec.*, 38(3):5–10, 2010.

[107] A. P. Pentland, R. W. Picard, and S. Scarloff. Photobook: tools for content-based manipulation of image databases, 1994.

[108] V. Pestov. Lower bounds on performance of metric tree indexing schemes for exact similarity search in high dimensions. *Algorithmica*, 66:310–328, 2013.

[109] W. Piekarski and B. Thomas. Arquake: The outdoor augmented reality gaming system. *Commun. ACM*, 45(1):36–38, 2002.

[110] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *SIGKDD*, pages 239–248. ACM, 2005.

[111] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *SIGKDD*, pages 570–579. ACM, 2007.

[112] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and nlp: Using locality sensitive hash function for high speed noun clustering. In *ACL*, pages 622–629. Association for Computational Linguistics, 2005.

[113] G. Reitmayr and D. Schmalstieg. *Collaborative augmented reality for outdoor navigation and information browsing*. TU?Wien, 2004.

[114] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453. Morgan Kaufmann Publishers Inc., 1995.

[115] S. Y. Rieh et al. Analysis of multiple query reformulations on the web: The interactive information retrieval context. *Information Processing & Management*, 42(3):751–768, 2006.

[116] J. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. *Advances in Spatial and Temporal Databases*, pages 205–222, 2011.

[117] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79. ACM, 1995.

[118] A. Roy, P. Mackin, J. Wallenius, J. Corner, M. Keith, G. Schymik, and H. Arora. An interactive search method based on user preferences. *Decision Analysis*, 5(4):203–229, 2008.

[119] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, 2008.

[120] H. Sagan. *Space-filling curves*, volume 2. Springer-Verlag, 1994.

[121] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[122] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, pages 743–754. ACM, 2004.

[123] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295. ACM, 2001.

[124] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5):430–441, 2012.

[125] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.

[126] M. Shokouhi and K. Radinsky. Time-sensitive query auto-completion. In *SIGIR*, pages 601–610. ACM, 2012.

[127] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[128] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *SIGGRAPH*, pages 835–846. ACM, 2006.

[129] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with. *PVLDB*, 8(1):1–12, 2014.

[130] D. R. Swanson. Information retrieval as a trial-and-error process. *The Library Quarterly*, pages 128–148, 1977.

[131] X. Tan, S. Chen, Z.-H. Zhou, and J. Liu. Face recognition under occlusions and variant expressions with partial similarity. *Information Forensics and Security, IEEE Transactions on*, 4(2):217–230, 2009.

[132] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3):20:1–20:46, 2010.

[133] A. K. H. Tung, R. Zhang, N. Koudas, and B. C. Ooi. Similarity search: a matching based approach. In *VLDB*, pages 631–642. VLDB Endowment, 2006.

[134] D. Van Krevelen and R. Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2), 2010.

[135] E. M. Voorhees. Query expansion using lexical-semantic relations. In *SIGIR94*, pages 61–69. Springer, 1994.

[136] H. Wang, S. Zhang, W. Liang, F. Wang, and Y. Yao. Content-based image retrieval using fractional distance metric. In *IASP*, pages 1–5, 2012.

[137] C. Xiao, W. Wang, and X. Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the VLDB Endowment*, 1(1):933–944, 2008.

[138] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492. IEEE, 2010.

[139] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 197–206. ACM, 2007.

[140] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, pages 545–556. IEEE, 2010.

[141] D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

[142] D. Zhang, B. C. Ooi, and A. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 688–699. IEEE, 2009.

[143] J. Zhang, W. Han, Y. Jia, and P. Zou. Interactive approach to eliciting users' preference using comparisons. In *ICCIS*, pages 497–500. IEEE, 2012.

[144] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130. ACM, 2008.

[145] M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[146] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD*, pages 915–926. ACM, 2010.

[147] Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung. Mesa: A map service to support fuzzy type-ahead search over geo-textual data. *Proc. VLDB Endow.*, 7(13):1545–1548, 2014.

[148] Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung. Inspire: A framework for incremental spatial prefix query relaxation. *TKDE*, 27(7):1949–1963, 2015.

[149] Y. Zheng, Q. Guo, A. K. H. Tung, and S. Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *SIGMOD*. ACM, 2016.

[150] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: Building a web-scale landmark recognition engine. In *CVPR*, pages 1085–1092. IEEE, 2009.

[151] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 694–707. springer, 2007.

[152] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162. ACM, 2005.

[153] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[154] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.