Geospatial data analysis : from querying to visualized exploration

Tao, Guo

2018

Tao, G. (2018). Geospatial data analysis : from querying to visualized exploration. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73327>

<https://doi.org/10.32657/10356/73327>

Geospatial Data Analysis: from Querying to Visualized Exploration

A thesis submitted
for a degree of Doctor of Philosophy
by

Guo Tao

Interdisciplinary Graduate School
Nanyang Technological University

August 2017

Acknowledgments

First and foremost, I would love to express my sincere gratitude towards my supervisor Dr. Gao Cong. Under his four-year elaborative supervision, I have not only learned the cutting-edge techniques, but also got a conscientious research attitude through critical thinking. He is not only a great supervisor for my research, but also a good friend in my life.

I would like to appreciate my friends in NTU. Xin Cao, Quan Yuan, Tuan Anh Nguyen, Shanshan Feng, Lisi Chen, Kaiyu Feng, Kaiqi Zhao, Zhida Chen, Daniel Rugeles Barbosa, Chuancong Gao, Jinpeng Wang, Zhen Hai, Yaqiong Liu, Xutao Li, Miao Li, Miao Xie, Wengen Li, Zongcheng Ji, Jilian Zhang, Seyed Ali Majid Zonoozi, Yiding Liu, Xiucheng Li, Guangtao Wang, and Shaokai Wang are being or have been supervised by Dr. Gao Cong. We have pleasant collaborations and/or great joy and fun together in the past four years. I thank them for the useful discussions and pleasant collaborations during my study from Dr. Lu Jiaheng and Dr. Zhifeng Bao. It has been a pleasure getting to know and work with them.

Contents

Acknowledgments	i
List of Figures	vii
List of Tables	ix
Summary	x
1 Introduction	1
1.1 Background and Motivations	1
1.2 Spatial Keywords Querying	2
1.2.1 Problems and Research Scope	2
1.2.2 Approaches and Methodology	4
1.3 Visualized Geospatial Data Exploration	5
1.3.1 Problems and Research Scope	5
1.3.2 Approaches and Methodology	9
1.4 Summary of Contributions	11
1.5 Thesis Organization	12
2 Literature Survey	13
2.1 Spatial Keyword Queries	13
2.1.1 Single Geo-textual Object Search	14
2.1.2 Group Geo-textual Objects Search	14
2.1.3 Other Geo-textual Objects Search	15
2.2 Geospatial Data Exploration	15
2.2.1 Cartographic generalization and selection	15
2.2.2 Interactive Data Exploration	17
2.3 Spatial Objects Simplification	18

2.3.1	Spatial Sampling	18
2.3.2	Query Result Diversification	18
3	Geo-textual Objects Querying	19
3.1	Problem and Background	19
3.1.1	Problem Statement	19
3.1.2	Existing Solutions for <i>mCK</i> Queries	20
3.2	Algorithm <i>GKG</i>	22
3.3	SKEC -Based Algorithms	24
3.3.1	Minimum Covering Circle and Keywords Enclosing Circle	24
3.3.2	Algorithm <i>SKEC</i>	27
3.3.3	Algorithm <i>SKECa</i>	30
3.3.4	Algorithm <i>SKECa+</i>	36
3.4	Exact Algorithm	38
3.4.1	Algorithm Framework	39
3.4.2	Procedure search()	41
3.5	Experiments	43
3.5.1	Experimental Settings	43
3.5.2	Experimental Results	44
3.6	Summary	54
4	Representative Geospatial Data Exploration	55
4.1	Problem Statement	55
4.1.1	Representative and Visibility Constraints	55
4.1.2	Advantages of the Representative Score	57
4.1.3	Spatial Object Selection (SOS) Problem	58
4.1.4	Interactive Exploration on Map	59
4.1.5	Interactive Spatial Object Selection (ISOS) Problem	60
4.2	Proposed Algorithm for SOS	61
4.2.1	A Greedy Algorithm	61
4.2.2	Approximation Ratio Analysis	65
4.3	Proposed Algorithm For ISOS	67

4.3.1	A Greedy Algorithm	68
4.3.2	Pre-fetching Strategy	68
4.4	A Sampling Extension	72
4.4.1	A Sampling Method	72
4.4.2	Determining Sampling Size	73
4.5	Experiments	74
4.5.1	Experimental Setup	74
4.5.2	A User Study on the Representative Score	76
4.5.3	Evaluation of Solutions to SOS Problem	78
4.5.4	Evaluation of solutions to ISOS Problem	83
4.6	Summary	86
5	Spatial Functional Region Exploration	87
5.1	Problem Statement	87
5.1.1	Road Segmentation and Spatial Objects	87
5.1.2	Information Loss	88
5.1.3	The FRS Problem	89
5.1.4	Convert FRS to a Graph Partitioning Problem	89
5.2	Approaches for FRS Problem	91
5.2.1	A Bottom-up Greedy Algorithm	91
5.2.2	From Greedy to Greedy ⁺	93
5.2.3	A Refinement Procedure	96
5.3	Experiments	97
5.3.1	Experimental Settings	98
5.3.2	Experimental Results	98
5.4	Summary	103
6	Conclusions and Future Work	104
6.1	Conclusions	104
6.2	Future Work	105
6.2.1	Extension of <i>mCK</i> Query	105
6.2.2	Time-aware Geospatial Objects Exploration	106
6.2.3	Diverse Geospatial Objects Exploration	106

A List of Publications	107
References	108

List of Figures

1.1	Example of the mCK query	3
1.2	Spatial Object Selection Example	6
1.3	Interactive Operations on Map	7
1.4	Example of crime distribution in a city	9
1.5	Functional Regions after segmentation and aggregation	10
3.1	Example of two diameters	25
3.2	Examples of enclosing circles w.r.t. $\{t_1, t_2, t_3, t_4\}$	26
3.3	Sweeping area	33
3.4	Computing two angles	33
3.5	Circle scan procedure w.r.t. query $\{t_1, t_2, t_3, t_4\}$	35
3.6	Varying ϵ	45
3.7	Varying number of query keywords	47
3.8	Comparing $SKEC$ with $SKECa+$	48
3.9	Varying optimal group diameter bound on LA	49
3.10	Varying optimal group diameter bound on TW	50
3.11	Varying timeout threshold	51
3.12	Varying query keywords frequencies	52
3.13	Varying ϵ	53
3.14	Scalability	54
4.1	Example for the greedy algorithm.	64
4.2	Status of the max-heap.	64
4.3	Conflict Objects Example	66
4.4	Zoom-in Example	70

4.5	Zoom-out Example	70
4.6	Panning Example	71
4.7	Selection Result by Different Methods	77
4.8	Comparing Different Methods on UK	79
4.9	Comparing Different Methods on POI	79
4.10	Varying ϵ on US	80
4.11	Varying δ on US	80
4.12	Varying query region size	81
4.13	Varying k	81
4.14	Varying distance threshold θ	82
4.15	Varying scalability on Twitter	82
4.16	Pre-fetching vs. non-fetching on UK	83
4.17	Varying query region size	84
4.18	Varying k	84
4.19	Varying distance threshold θ	85
4.20	Varying scalability	85
4.21	Varying zooming scale and panning overlap on UK	86
5.1	Example of Graph Conversion	90
5.2	Example of Graph Shrinking	92
5.3	Example of Graph ⁺	94
5.4	Example of Refinement	96
5.5	Comparing Algorithms on London	99
5.6	Comparing Algorithms on Beijing	99
5.7	Comparing Algorithms on Singapore	100
5.8	Varying k on London	100
5.9	Varying k on Beijing	100
5.10	Varying k on Singapore	101
5.11	Varying d on London	101
5.12	Varying d on Beijing	102
5.13	Varying d on Singapore	102
5.14	Scalability on London	102

List of Tables

2.1	Comparison of closely related work on different aspects	15
3.1	Dataset properties	44
4.1	Parameters	76
4.2	User Study Result for SOS	78
4.3	User Study Result for ISOS	78
5.1	Statistics of the dataset	98

Summary

With the proliferation of social sharing platforms (e.g., Foursquare and Yelp) and online social media (e.g., Twitter, Facebook and Instagram), large collections of geospatial data are becoming available, such as geo-tagged photos and geo-textual posts. The availability of substantial amount of such **geospatial objects** gives prominence to the spatial keyword query, which is to find the geo-textual objects that best match the query arguments exploiting both locations and textual descriptions. As an important type of spatial keyword query, the m -closest keywords (mCK) query is useful in many applications such as detecting locations of web resources. However, the existing work does not study the intractability of this problem and only provides exact algorithms, which are computationally expensive.

Since the volume of geospatial data keeps growing rapidly, an emerging challenge is how to explore and understand the data before we analyze it. Presenting geospatial data through a visualization system is no doubt the most efficient way for end users, however, a remaining challenge is how to read the oversize data effectively. We aim at solving the challenge from two perspectives. On the one hand, the users may be exhausted and distracted if too many results are displayed at the same time. Following the rule of “Less is more”, we can show a small set of representative objects to users instead of the whole collection. Thus, the challenge in this problem is how to build such a map rendering system that efficiently selects a small set of representative objects from the current region of user’s interest. On the other hand, as the characteristics of geo-spatial data, the objects of similar types or functions may tend to appear together. Such circumstance is common in real-life, for example, there are more shopping malls grouping in the downtown area of a city than the living zones. To this end, we can explore the geo-spatial data by dividing the whole space into several functional regions according to the utilities of the geo-spatial objects.

First, we study the m -closest keywords (mCK) query, finding a group of objects such that they cover all query keywords and have the smallest diameter, which is defined as the largest distance between any pair of objects in the group. We prove that the problem of answering mCK queries is NP-hard. We first devise a greedy algorithm that has an approximation ratio of 2. Then, we observe that an mCK query can be approximately answered by finding the circle with the smallest diameter that encloses a group of objects together covering all query keywords. We prove that the group enclosed in the circle can

---

answer the mCK query with an approximation ratio of $\frac{2}{\sqrt{3}}$. Based on this, we develop an algorithm for finding such a circle exactly, which has a high time complexity. To improve efficiency, we propose another two algorithms that find such a circle approximately, with a ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$. Finally, we propose an exact algorithm that utilizes the group found by the $(\frac{2}{\sqrt{3}} + \epsilon)$ -approximation algorithm to obtain the optimal group. We conduct extensive experiments using real-life datasets. The experimental results offer insights into both efficiency and accuracy of the proposed approximation algorithms, and the results also demonstrate that our exact algorithm outperforms the best known algorithm by an order of magnitude.

Next, we study how to develop an interactive visualization map exploration system. We propose that such system should support the following desirable features: representativeness, visibility constraint, zooming consistency, and panning consistency. The first two constraints are fundamental challenges to a map exploration system, which aims to efficiently select a small set of representative objects from the current region of user’s interest, and any two selected objects should not be too close to each other for users to distinguish in the limited space of a screen. We formalize it as the Spatial Object Selection (SOS) problem, prove that it is an NP-hard problem, and develop a novel approximation algorithm with performance guarantees. To further support *interactive* exploration of geospatial data on maps, we propose the Interactive SOS (ISOS) problem, in which we enrich the SOS problem with the zooming consistency and panning consistency constraints. The objective of ISOS is to provide seamless experience for end-users to interactively explore the data by navigating the map. We extend our algorithm for the SOS problem to solve the ISOS problem, and propose a new strategy based on pre-fetching to significantly enhance the efficiency. Finally we have conducted extensive experiments to show the efficiency and scalability of our approach.

Last but not least, we study how to partition the geospatial objects into functional regions according to the utilities and spatial distributions. It aims to aggregate similar and adjacent objects into enclosed regions that indicate certain functions. During this process, since the attributes distribution of the geospatial objects are aggregated, some information is lost during this process. We define the *information loss* to measure how much information is lost when merging two sets of objects. To reduce the possibilities of generated regions, we exploit the existing road networks to limit the boundaries of the separated regions to be roads, since it is the natural partition of cities and people live in these roads-segmented regions and POIs (points of interests) fall in these regions. We formulate this problem as Functional Region Segmentation (FRS) problem, and prove that it is an NP-hard problem. We develop a bottom-up greedy algorithm to solve the FRS problem, which terminates in limited steps. Results of empirical studies show that our proposed algorithm is able to solve FRS problem efficiently and effectively.

Chapter 1

Introduction

In this chapter, we first introduce the background and motivations of geospatial objects analysis in Section 1.1. Next, in Section 1.2 we present research problems and methodologies for Spatial Keywords Querying problems. In Section 1.3 we discuss the research problems and methodologies for Visualized Geospatial Data Exploration problems. We summarize our contributions of the thesis in Section 1.4 and state the thesis organization in Section 1.5.

1.1 Background and Motivations

With the proliferation of GPS-equipped mobile devices, massive amounts of geospatial objects are becoming available on the web that each possess both geographical content and other content, such as texts and photos. For example, such geospatial objects include points of interest (POIs) associated with texts, such as tourist attractions, hotels, restaurants, businesses, entertainment services, etc. Other example geospatial objects include geo-tagged micro-blogs (e.g., Tweets), photos with both tags and geo-locations in social photo sharing websites (e.g., Flickr), and check-in information on places in location-based social networks (e.g., FourSquare).

The availability of substantial amount of geo-textual objects gives prominence to the **spatial keyword queries** that target these objects, which have been studied extensively in recent years [DFHR08, ZCM⁺09, CJW09, ZZZL13, ZTT13, CCJW13]. Typically, a spatial keyword query finds the objects that best match the arguments in the query exploiting both locations and textual descriptions. Such queries are widely used in many services and applications, such as online Map services, travel itinerary planning, etc.

Since large collections of geospatial data are becoming increasingly available in the big data era, a rising problem is how to understand the data before we decide to use the data for different applications. It involves how to read and understand the data from different aspects. For geospatial data, it is useful to provide support for end-users or data scientists to perform **visualized exploration** on maps. Plotting the geospatial objects on a map is a possible way to visualize the distribution of data. Users can zoom-in or zoom-out the map to explore the data from different scopes. However, it only exploits the geographical information and other content of the geospatial objects, such as texts, is not reflected on the map.

One way to improve the experience of exploration is providing a few examples from the data and displaying them on a map to users. However, if too many samples are shown on a single map it is still hard for users to read and obtain valuable information. In addition, if the samples are similar to each other and redundant, it can not reflect what the original data really is. Alternatively, a possible solution is, we can offer the users a desired number of samples from the whole collection and they can represent most content of the whole dataset. Since the size of geospatial objects is typically very large, there are huge number of possibilities to select even very few number of sample, thus how to choose a limited subset of samples that can best represent the original dataset is in our research scope.

Another way to provide informative visualization is dividing the geospatial objects into functional regions on a map. The motivation behind is that the spatial objects are usually not randomly distributed in the space, however, similar types of objects may tend to appear together. For example, a city may be partitioned into functional regions such as living zones, commercial centers, entertainment regions, industry regions, and etc. In the commercial center, it is possible to find more shopping malls than in a living zone. The functional region segmentation provides a view of the distribution of the geospatial data from the perspective of both geographical adjacency and content connectivity.

1.2 Spatial Keywords Querying

1.2.1 Problems and Research Scope

As an important type of the spatial keyword query, the m -closest keywords (mCK) query [ZCM⁺09, ZOT10] is defined for finding a set of closest keywords in the geo-textual object database. Specifically, let $\mathcal{O}$ be a set of geo-textual objects, and each

object $o \in \mathcal{O}$ has a location denoted by $o.\lambda$ and a textual description $o.\psi$. The mCK query q contains m keywords, and it finds a group of objects G such that they cover all the query keywords (i.e., $q \subseteq \bigcup_{o \in G} o.\psi$) and such that the diameter of this group, denoted by $\delta(G)$, is minimized. The diameter of a group is defined as the maximum distance between any pair of objects in the group.

Figure 1.1: Example of the mCK query

The mCK query has many applications as shown in the proposals [ZCM⁺09, ZOT10]. For example, it can be used in detecting geographic locations of web resources such as documents or photos. Given a document or a photo with some tags, we can issue an mCK query using these tags. After a group of objects covering all tags that have the smallest diameter is found, the area where these objects locate is very likely to be the location of the document or the photo. It has been shown [ZCM⁺09, ZOT10] that this approach can be used to effectively address the challenge faced by the traditional location detection techniques when the tags of documents or photos do not contain gazetteer terms. The mCK query also has potential applications for location-based service providers [ZOT10]. One example application is “fans of Apple products can submit ‘Apple store subway’ to locate a retailer store near the subway for convenient purchase of the products in New York [ZOT10].” As another example, consider a tourist who is planning for a trip to Kyoto. She wishes to explore an area where the following attractions are within walking distance (i.e., close to one another): *shrine*, *shop*, *restaurant*, and *hotel*. That is, she is seeking a location to stay and do sightseeing and shopping on foot. This can be formulated as an mCK query. As shown in Figure 1.1, we can perform search on objects within the area of Kyoto, and the group enclosed in the circle is returned to meet the tourist’s requirements.

1.2.2 Approaches and Methodology

Exact algorithms are proposed in the studies [ZCM⁺09, ZOT10] that run exponentially with the number of objects relevant to the query, and thus they are computationally prohibitive when the number of relevant objects is large. For example, in our experiments, the best known algorithm [ZOT10] took almost 1 hour to answer a query containing 8 keywords on a dataset with 1 million objects. Moreover, the hardness of the problem is still unknown.

In this thesis, we establish that the problem of exactly answering the mCK query is NP-hard, which can be proven by a reduction from the 3-SAT problem. The intractability result motivates us to design approximation algorithms for efficiently processing the mCK query. We first develop a greedy approach that has an approximation ratio of 2. We call this algorithm the Greedy Keywords Group algorithm, denoted by GKG . Utilizing the result returned by GKG , we propose three non-trivial approximation algorithms that all have better performance guarantee than GKG . Based on one of them, we further develop an efficient exact algorithm.

The three approximation algorithms answer the mCK query by finding the circle with the smallest diameter that encloses a group of objects together covering all query keywords. We call such a circle the “smallest keywords enclosing circle,” and given a query q , we denote the circle by $SKEC_q$. We prove that the group in $SKEC_q$ can answer the mCK query with an approximation ratio of $\frac{2}{\sqrt{3}}$. Although finding $SKEC_q$ is solvable in polynomial-time, it is still an open problem to find $SKEC_q$ efficiently, which is challenging because we know neither its radius nor its center.

We first develop an approach to finding $SKEC_q$ exactly. We denote the set of objects that contain at least one query keyword by $\mathcal{O}'$. This algorithm is based on a lemma we establish: there must exist either three or two objects in $\mathcal{O}'$ on the boundary of $SKEC_q$ and they determine $SKEC_q$. Unfortunately, this method has a high time complexity ($|\mathcal{O}'|^4$ in the worst case). We call this method the Smallest Keywords Enclosing Circle (denoted by $SKEC$, with a bit abuse of notation). This method is impractical when $|\mathcal{O}'|$ is large.

For better efficiency, we propose to find $SKEC_q$ approximately. First, we develop an algorithm that performs search on each object o in $\mathcal{O}'$ one by one. We call this method the Approximate Smallest Keywords Enclosing Circle (denoted by $SKECa$) algorithm. We prove that the group enclosed in the circle found by $SKECa$ can answer the mCK query with an approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ (ϵ is an arbitrarily small positive value). To further improve efficiency, we devise techniques to perform the search on all objects in

$\mathcal{O}'$ together, instead of on each of them separately. This algorithm finds the same circle as found by *SKECa*. We denote this enhanced algorithm by *SKECa+*. Both algorithms have better time complexity than that of *SKEC*.

Finally, based on *SKECa+*, we devise an exact algorithm for the *mCK* query. Since answering *mCK* queries is NP-hard, it is challenging to devise an efficient exact algorithm—an exhaustive search on the object space cannot be avoided. We prove that the diameter of the circle that encloses the optimal group cannot exceed $\frac{2}{\sqrt{3}}$ times the diameter of the group found in *SKECa+*. Thus, we do an exhaustive search around each object o in $\mathcal{O}'$, considering only objects in $\mathcal{O}'$ whose distances to o are smaller than $\frac{2}{\sqrt{3}}$ times the diameter of group G returned by *SKECa+*. We denote this method by *EXACT*. The group G found by *SKECa+* is able to greatly reduce the search space of *EXACT*.

1.3 Visualized Geospatial Data Exploration

1.3.1 Problems and Research Scope

1.3.1.1 Representative Geospatial Data Exploration

It is useful to provide support for end-users to perform visualized exploration on such geospatial data on maps. We illustrate the desired features of such systems in the following example.

Example 1: Given a collection of points of interest (POIs), an end-user would like to browse a small number (denoted by k) of representative POIs for an area on an online map using her pad. Ideally, the set of selected POIs can well represent the POIs of the area (i.e., Representative Constraint), and they should not be too close to each other so that they will not overlap with each other on the map (i.e., Visibility Constraint) when shown on the pad screen. Figure 1.2 demonstrates Example 1, where a small number of representative POIs are shown to users (Figure 1.2(b)). The user may be interested in some POIs, and he may click one to check more information such as descriptions and user’s voting score. Moreover, the “hidden” POIs that are represented by the user-viewing POI are also highlighted (Figure 1.2(c)) as related recommendations for users to explore. □

Goal 1: Efficient Spatial Object Selection (sos)

Given the current region of user’s interest, how to efficiently select a set S of k objects

Figure 1.2: Spatial Object Selection Example

(among all the spatial objects falling in this region), so that it meets (i) Visibility Constraint — the distance between any two objects in S is larger than a given distance threshold θ , and (ii) Representative Constraint — the aggregated similarity between S and the whole spatial objects in that region is maximized. Note that in addition to spatial distance, we can also include other factors in defining the visibility constraint, such as the semantic similarity of two objects.

By reviewing the literature in the areas of cartographic selection and spatial sampling (see Table 2.1), we find that there have been some studies [KVSZ14, NAS12, PSA14] taking the visibility constraint into account. However, none of them considers representativeness except for a study done by Drosou et al. [DP12], in which it is assumed that

the representativeness of a spatial object is based on its spatial distance to other objects, and its proposed solution is built based on the assumption.

However, for a truly general solution, the representativeness/similarity definition should be application dependent. For instance, in Example 1, we could consider both the distance of two POIs and the semantic similarity of the two POIs. Furthermore, if additional information, such as the popularity of POI and the importance of tweets, is available, it should be taken into account when selecting the set of representative objects.

To this end, we propose the SOS problem which for the first time takes both visibility constraint and representative constraint into consideration in object selection, and we accept any similarity metric in defining visibility constraint and representative constraint.

Figure 1.3: Interactive Operations on Map

Goal 2: Interactive Spatial Object Selection (isos)

The SOS problem deals with the object selection problem at the current region of user's interest. To support the objective of exploring a geospatial dataset, we should provide the interactive exploration function so that users can interact with the map by performing map navigation operations, including zoom-in, zoom-out and panning. As users navigate the map, the proposed visualized exploration system needs to maintain consistency when selecting a new set of representative objects for the new map region, as illustrated in the following example.

Example 2: Continue with Example 1. The user now would like to further explore the POI dataset by zooming-in on the map as illustrated in Figure 1.3(a)). There are 9 POIs in the regions, denoted by $o_1, \dots, o_9$. The black nodes are visible to the user. Those spatial objects that are visible in the current granularity and fall in the new map region should also be visible in the new map region at a finer granularity. As a specific example, object o_5 should be visible after zooming in. In addition, objects o_3 and o_4 become visible after zooming in. Similarly, when zooming out, the set of spatial objects that are visible in the coarser granularity should be a subset of the spatial objects that are visible in a finer granularity as illustrated in Figure 1.3(b). When the user pans the map from the current window, denoted by r_1 , to a nearby region r_2 that overlaps with r_1 , the objects appearing in the overlapping area before the movement should also appear in the region after the movement. As shown in Figure 1.3(c), objects o_1 and o_5 are still visible after the panning. $\square$

The consistency constraint with respect to zoom-in and zoom-out operations is referred to as *zooming consistency*, and the consistency constraint with respect to panning operation is referred to as *panning consistency*. Based on the SOS problem, the ISOS problem additionally considers the zooming consistency and panning consistency to provide the seamless experience for end-users to interactively navigate a map to explore the data.

1.3.1.2 Functional Region Exploration

To provide a deep insight of the geospatial objects distribution w.r.t. the associated content, we can provide a visualized exploration of functional regions on a map. We illustrate the features and usabilities of such visualization system in the following examples.

Consider the map of a city shown in Figure 1.4, where there are three roads partitioning the city into 7 blocks (although in real-world there could be tens of thousands of blocks). We would like to analyze the three types of crimes happened in this city during the last 5 years, which could be *theft*, *assault*, and *robbery*. For each piece of crime record (as a special type of geospatial object), we extract the three columns of information: the location where the crime happened, the type of crime, and the dangerous level of it. In each block we can calculate the percentage of each crime and they are shown in Figure 1.4 as pie charts.

However, this low-level visualization is not directly helpful for analysis and clear enough, since too many details are shown on the map and the users can not obtain useful

Figure 1.4: Example of crime distribution in a city

information from it. To provide a better high-level summarization of the geospatial objects, the system aggregates the blocks of similar distribution of objects into some *functional regions*, where the information in each functional region is aggregated from low-level blocks. Figure 1.5 demonstrates an aggregation result of 2 functional regions.

Applying the crime data on such functional region exploration system is useful for the study of public security, and it can be easily extended to other applications. For example, in a politics campaign, we can analyze the distribution of election support rate for different parties. By applying the POI (Point of Interest) data on such system, we can analyze the functional components of a city, e.g. living zones, commercial centers, entertainment regions and industry regions, for city planning and development.

1.3.2 Approaches and Methodology

1.3.2.1 Representative Geospatial Data Exploration

We formally introduce four desired features that an interactive visualized exploration system on a map should meet, i.e., representative constraint, visibility constraint, zooming consistency constraint and panning consistency constraint. To the best of our knowledge, this is the first work that is able to meet all the four features to achieve an interactive visualization map exploration system.

Figure 1.5: Functional Regions after segmentation and aggregation

We propose the problem of Spatial Object Selection (SOS) considering representative constraint and visibility constraint, and prove that it is an NP-hard problem. The intractability result motivates us to design approximation algorithms for the SOS problem. We propose a greedy algorithm with performance guarantees and it has an approximation ratio of $1/8$.

We propose the problem of Interactive Spatial Object Selection (ISOS) by enforcing the zooming consistency constraint and panning consistency constraint on top of the SOS problem to support interactive exploration of a geospatial data on maps. We extend the proposed greedy algorithm to address the ISOS problem. We further propose a new approach to improve the efficiency of our algorithm in finding the new set of representative objects to support interactive exploration in ISOS, by up to 2 orders of magnitude (Section 4.3).

To improve the efficiency of SOS and ISOS on large datasets, we propose to use sampling strategy (Section 4.4). We prove that it has a theoretical guarantee, and the experiments show that it only needs less than 2% of the whole data (which is of size up to 100 million) to achieve a very small error bound (Section 4.5.3.2).

We have conducted extensive experiments over three real-life datasets to evaluate the performance of our proposed algorithms (Section 4.5). The experimental result shows that our greedy algorithm is efficient. Our pre-fetching technique further improves the efficiency of our greedy algorithm for ISOS by almost 2 orders of magnitude, and it

ensures a very high response speed (usually 0.1 second) for new object selection to cater for zoom-in, zoom-out and panning operations.

1.3.2.2 Functional Region Exploration

We first formally define the Functional Region Segmentation (FRS) problem. By defining the concept of *information loss*, we measure the quality of segmentation and aggregation. To reduce the number of possibilities of segmented regions, we propose to use existing road network as the segmentation boundaries.

We demonstrate that the FRS problem can be converted to a set partitioning problem on a graph and prove that the FRS problem is NP-hard.

Due to the hardness of the problem, we design a greedy approach for the FRS problem. We organize the graph as a hierarchy structure and the greedy approach shrinks the size of graph in a bottom-up way. We show that in $O(n)$ steps the greedy approach terminates. To further improve the efficiency, we propose an accelerating strategy that shrink the size of graph by half in each round of iteration. We show that the enhanced approach terminates in only $O(\log(n))$ steps.

1.4 Summary of Contributions

In summary, the main contributions of this thesis are stated as follows.

- First, we prove that the problem of answering mCK queries is NP-hard. We observe and prove that an mCK query q can be answered by $SKEC_q$ approximately with a performance guarantee. Based on this, we present novel approximation algorithms, $SKEC$, $SKECa$, and $SKECa+$, all with provable performance bounds for answering the mCK query. We also design an exact approach based on $SKECa+$ with the worst case time complexity $O(|\mathcal{O}'|n^{|q|-1})$, where n is the number of objects in a region around an object, which can be bounded by the result of $SKECa+$, and in general $n \ll \mathcal{O}'$. Its complexity is better than that of the best known algorithm [ZOT10], which is $O(|\mathcal{O}'|^{|q|})$. We conduct extensive experiments on real-life datasets. The experimental results demonstrate that the proposed approximation algorithms offer scalability and excellent efficiency and accuracy and that the exact algorithm outperforms the best known algorithm for answering mCK [ZOT10] by orders of magnitude.

- Second, we prove that our problem of selecting the most representative geospatial objects is unfortunately NP-hard. We design a greedy algorithm to select a set of representative geospatial objectives from a large number of geospatial objects, while the selected geospatial objects should not be too close with each other to ensure each selected object is visible on the map. We prove that the greedy algorithm has an approximation ratio of $1/8$. Furthermore, in order to further improve the efficiency of the objects selection, we propose to use a few sampled objects that can guarantee a certain error tolerance with high probability. In addition, we extend the proposed greedy algorithm to handle the zooming consistency and movement consistency to find new set of representative objects when user navigate the map. We further propose a new approach to leverage prefetching results to improve the efficiency of our algorithm in finding the new set of representative objects to support interactive exploration.
- Third, we define the FRS problem that partitions the space into functional regions with minimum information loss. We show that the problem can be converted to a set partitioning problem on graph, and prove that it is NP-hard to be solved. We design a greedy algorithm that organizing the space from bottom to top, and it is guaranteed to terminate in limited steps. We propose an enhanced greedy algorithm with improved time complexity, and we design a post-processing algorithm to further improve the performance based on a valid result.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 summarizes the related work for spatial keyword queries, geospatial data exploration and spatial objects simplification. In Chapter 3, we present the problem of the m -closet keyword query and solutions. Next, in Chapter 4 we introduce the SOS problem for selecting representative geospatial data and the extension ISOS problem for interactive geospatial data exploration. In Chapter 5, we discuss the geospatial data exploration for functional regions. Finally, in Chapter 6, we conclude the thesis and discuss the available orientations of future work.

Chapter 2

Literature Survey

My research is related to spatial keywords querying and geo-spatial data exploration. In this chapter, we first review the literature that studies the three branches of spatial keyword queries in Section 2.1. The work of processing *mCK* queries extends the studies introduced in Section 2.1.2. Then, in Section 2.2 we review the related work of geospatial data exploration. We compare our work with existing approaches to support interactive visualized exploration from different aspects. Finally, in Section 2.3 we review two families of related work that also aim to reduce the size of data for better user exploration experience.

2.1 Spatial Keyword Queries

Geo-textual object contains both a geographical location and a textual description are gaining in prevalence. As discussed in Chapter 1, they can be obtained from commercial map services (e.g., Google Maps) or social networks (e.g., Instagram and Foursquare). As the growing volume of geo-textual objects, classic researches focus on how to retrieve these objects efficiently. Existing works study how to efficiently answer the queries from the users for different needs. In this thesis, we study the type of queries that contain a geographical location which could be the user's current location and a set of keywords describing the user's particular need. We categorize and discuss the geo-textual object search into three types according to their utilities. In Section 2.1.1 we will introduce the basic type of query, and in Section 2.1.2 we will discuss the group objects search which are typically complex due to the number of combinations. Finally in Section 2.1.3 we will discuss some other types of spatial keyword queries.

2.1.1 Single Geo-textual Object Search

The spatial keyword queries are gaining in prevalence and are widely used in many real-life applications. For example, in Google Maps “search nearby” offers users the functionality to retrieve points of interest around a specified location. Most existing studies on spatial keyword querying focus on retrieving a list of single geo-textual objects such that each object returned is both relevant to query keywords and close to query location. The spatial keyword queries can be categorized into two types according to how keywords are used in querying. In some proposals (e.g., [HHLM07, DFHR08, CWR10, CHD⁺11, ZTT13, ZZZL13]), keywords are used as Boolean predicates to filter out objects that do not contain the keywords, and the remaining objects are ranked based on their spatial proximity to the query. In other proposals (e.g., [CJW09, CHD⁺11, LLZ⁺11]), spatial proximity and textual relevance are combined by a linear function to rank geo-textual objects.

2.1.2 Group Geo-textual Objects Search

Several recent proposals consider searching for a group of geo-textual objects instead of single objects. As described in Section 3.1.2, the *mCK* query is proposed and studied in the works [ZCM⁺09, ZOT10]. The existing studies on the *mCK* query [ZCM⁺09, ZOT10] do not establish the intractability of this problem and propose exact algorithms only. We prove that answering the *mCK* query is NP-hard and propose both exact and approximation algorithms. The spatial group keyword query SGK [CCJO11, LWWF13] takes a location and a set of keywords as query arguments. It retrieves a group that covers all the query keywords, has a small diameter, and is close to the query location. As analyzed in Section 3.1.2, one special case of the SGK query [CCJO11] is equivalent to the *mCK* query, and the algorithm proposed is reduced to the method [ZCM⁺09] when being applied to the *mCK* query, and thus is worse than the baseline *VirbR* [ZOT10], used in our experiments. The approximate algorithm [CCJO11] performs worse than our algorithm *GKG* as shown in our experiments. As discussed in Section 3.1.2, another type of the SGK query [LWWF13] can be adapted to answer the *mCK* query, and as shown in the experimental study, its performance is much worse than our proposed exact algorithm. The SGK query and the *mCK* query suit different application scenarios. Consider the tourist example in Section 1.2.1. If the tourist has already booked a hotel, then the SGK query could be used where the hotel serves as the query location. Otherwise, the *mCK* query is suitable where “hotel” is one of the query keywords. Long et al. [LWWF13] also

Table 2.1: Comparison of closely related work on different aspects

Methods	Zooming Consistency	Movement Consistency	Visibility Constraint	Representativeness	Similarity Metrics	Online
Our Work	Yes	Yes	Yes	Yes	General	Yes
Sarma et al. [DSLG ⁺ 12]	Yes	No	No	No	N.A.	No
Konstantin et al. [KVSZ14]	Yes	No	Yes	No	N.A.	No
DisC [DP12]	No	No	Yes	Yes	Distance	Yes
Nutanong et al [NAS12]	No	Yes	Yes	No	N.A.	Yes
Mahdian et al. [MSV15]	No	No	No	No	N.A.	Yes
Peng et al. [PSA14]	Yes	No	Yes	No	N.A.	Yes

study the spatial group keyword query, and they design a more efficient exact algorithm for the problem.

2.1.3 Other Geo-textual Objects Search

In addition, some proposals consider spatial keyword queries on road networks (e.g., [RJN12]), and spatial keyword queries on trajectory databases are also studied (e.g., [SDY⁺12, ZSYY13]). There also exists some work (e.g., [FLZ⁺12]) querying geo-textual objects whose locations are represented by rectangles. The problem of spatio-textual similarity joins is also studied to join two sets of geo-textual objects [BGM12, LLF12]. Finally, Kargar et al. [KA11] study the problem of finding the r -clique with minimum weight, which is different from mCK , as the diameter is given as a query parameter in the work [KA11] while mCK aims to minimize the diameter of a group.

2.2 Geospatial Data Exploration

Although visualization of geographical data, such as POIs, is an important part of online map systems such as Google Maps and MapQuest, we believe that our study of SOS problem is the first work to study the geospatial object selection problem that takes into account the representativeness of objects to support interactive visualized exploration. Table 2.1 summarizes the difference between our work of SOS problem and closely related work in terms of functions, similarity metrics adopted and whether online computation is supported for interactive exploration.

2.2.1 Cartographic generalization and selection

Our work is related to cartographic generalization [FT94, PD95, SM89, WJT03], which is a classic research topic dealing with the selection and transformation of geographic

information at a high level of detail into information at a lower level detail rendered on a map. One of the challenges in cartographic generalization is sifting through all of the data available and deciding what to place on the map at a given zoom level. We next review the recent studies addressing the challenge.

Sarma et al. study the map thinning problem [DSL⁺12], which is to determine appropriate samples of data to be shown on each pre-defined geographical region and zoom level on maps. The whole world map is represented by a quadtree and each zoom level corresponds to a level of the quadtree. The objective is to select the maximal number of geospatial objects for each cell region while the following three constraints are satisfied: 1) The visibility constraint is set by selecting a given number of geospatial objects for each cell region; 2) across different zoom levels the zoom consistency constraint should be satisfied; and 3) each polygon object is visible in its entirety when moving a map around at the same zoom level. From the solution perspective, Sarma et al. [DSL⁺12] presents an integer programming formulation to solve the problem and a more efficient solution based on DFS traversal of a spatial tree is also proposed. The objective criteria are simple: either show the maximum number of records or show the objects with high scores. Konstantin et al. [KVSZ14] extend the work [DSL⁺12] by proposing a concept similar to the visibility constraint defined in our work. Our work differs from the work [KVSZ14, DSL⁺12] in the following two aspects. First, Pre-defined Granularity & Region Cell vs. Arbitrary Granularity & Region. The approaches [KVSZ14, DSL⁺12] are precomputation-based, and perform an *offline* computation of selecting objects for all cells of different zoom levels. However, in real-world a user's region of interest may not match well with any of the pre-defined cells, and thus the selected objects for those pre-defined cells may not provide a good solution to our SOS problem. When we have to use the pre-computed results from several cells overlapping with the user region to select objects for the region, it is possible that none of the selected objects in these cells fall in the user region. In contrast, we provide an efficient online solution to selecting objects catering for any region of interest at an arbitrary granularity. Moreover, the pre-selected objects will not work if users specify some filtering condition, e.g., names should contain "restaurant." Second, our work considers the representative constraint while the previous ones do not.

Nutanong et al. [NAS12] define the problem of sampling large geographic dataset falling within a region of user interest for display, and they propose to utilize SQL statements to achieve several features like visibility constraint and object distinctiveness

(representing importance of object). However, it does not explicitly address the zooming consistency and panning consistency constraints. Instead, it claims that distinctiveness score of each object does not change when panning and will increase when zooming in. Similar to our work, the work is an online solution and supports filtering conditions on the dataset. It optimizes a *Distinctiveness Score* to maximize the diversity of selected objects, which is based on an intuition that if an object is far from the others then it should be assigned with a high score and more likely to be selected. In general, the objectives in these works all ignore the content of the spatial records, and thus the sampled results are lack of diversity. Besides, all these studies suppose the objects are selected offline, and the sampling results for each user are the same. In our approach, however, we also solve the online queries, where only the objects that satisfy the query constraints can be the selection candidates. The online selection problem is personalized and challenging, since the user can issue any type of query and we have to guarantee the efficiency of our algorithm such that the results can be displayed in real time.

Mahdian et al. [MSV15] propose a POI selection problem, which aims to highlight a subset of POIs with the maximum utility instead of showing all to users. The utility of each POI is a user-defined function, and it takes into account the POIs relevance and quality. This is similar to the weight in our problem definition. The problems in the two studies are different from our problem as our work considers the representativeness of geospatial objects and the visibility constraints. Furthermore, we explicitly address the consistency constraint to support visualized exploration as users navigate a map.

The map labeling problem [CMS95] is to select a subset from a geographic dataset to maximize the sum of their importance without any overlap between the textual labels of selected objects. Peng et al. [PSA14] develop a filter-and-refine algorithm for the problem. The map labeling problem is different from our work in that it does not consider the representativeness of objects.

2.2.2 Interactive Data Exploration

Our work is related to studies on interactive exploration of geospatial data. The existing work supporting Interactive Data Exploration [IPC15] is query-based, where users interactively issue multiple queries until the user is satisfied with the result. However, these approaches have different focus from our work that aims to select representative geospatial objects. For example, ScalaR [BSC13] is a system that dynamically performs resolution reduction by inserting aggregation, sampling or filtering operations to reduce

the size of the result when the expected result of a DBMS query is too large to be effectively rendered.

2.3 Spatial Objects Simplification

2.3.1 Spatial Sampling

Our work is related to sampling techniques, which are widely used in many applications [Coc07, KBP⁺15]. Recently, Wang et al. [WCLY15] consider the spatial sampling problem, in which some objects are first randomly selected for a spatial range query and the number of objects is repeatedly increased until the user terminates the query. The objective is for statistical aggregations such as calculating the average price of all sale transactions in NYC based on a small sample of data. Our problem and objective are different from those for spatial sampling.

2.3.2 Query Result Diversification

Query result diversity has been extensively studied to improve the user's experience (e.g., [AK11, DP12, QYC12, FMT12]). Our work is related to DisC [DP12], which aims to select a subset of diversified spatial objects that can represent the whole dataset. Specifically, DisC imposes two constraints: (1) Distance. The distance between any pair of objects in the DisC set must be larger than r , where r is a user-given parameter. Note that the parameter r can also be used to tune the number of selected objects. (2) Coverage. The DisC set S can cover an object o , if there exists an object $o' \in S$ and the distance between o and o' is less than r . The objective of DisC is to find a DisC set satisfying the distance constraint to cover the maximum number of objects. The representativeness of this work is based on spatial coverage, that is, each object should be covered by a circle of radius r centered at one object in the selected subset. Our problem is more general than DisC since we consider the representativeness, which can be measured based on any attribute of the geospatial objects, e.g., textual similarity. In contrast, in DisC the representativeness of an object is based on its spatial distance to other objects. However, the algorithm for solving the DisC problem is dependent on the distance metric, and is not applicable to a general definition of representativeness based on any similarity function. Furthermore, we consider the consistency constraint to support interactive exploration, which is not considered in DisC.

Chapter 3

Geo-textual Objects Querying

This chapter is organized as follows: Section 3.1 formally defines the *mCK* problem. Section 3.2 introduces the greedy algorithm that we proposed and shows the approximation ratio. We introduce our proposed approximation algorithm *SKECa+* in Section 3.3, then in Section 3.4 we extend to the exact algorithm *EXACT*. Section 3.5 presents the experiments of all our proposed algorithms. Finally we summarize our contributions of this chapter in Section 3.6.

3.1 Problem and Background

3.1.1 Problem Statement

Let $\mathcal{O}$ be a database consisting of a set of geo-textual objects. Each object $o \in \mathcal{O}$ is associated with a location $o.\lambda$ and a set of keywords $o.\psi$ describing the object (e.g., the menu of restaurants).

Definition 1: Diameter of a group: Given a group of objects G , its diameter is defined as the maximum Euclidean distance between any pair of objects in G , denoted by $\delta(G)$. That is, $\delta(G) = \max_{o_i, o_j \in G} \text{Dist}(o_i, o_j)$, where $\text{Dist}(o_i, o_j)$ computes the Euclidean distance between o_i and o_j . $\square$

Definition 2: Problem definition [ZCM⁺09, ZOT10]: An *m-closest keywords* (*mCK*) query q contains m keywords $\{t_{q_1}, t_{q_2}, \dots, t_{q_m}\}$, and it finds a group of objects $G \subseteq \mathcal{O}$, each containing at least one query keyword, such that $\cup_{o \in G} o.\psi \supseteq q$ and such that $\delta(G)$ is minimized. $\square$

Definition 3: Feasible group: Given an *mCK* query q , if a group of objects can cover all keywords in q , we call such a group a “feasible group” or a “feasible solution”. $\square$

We establish the hardness of answering the mCK query by the following theorem. Note that the hardness of answering the mCK query is not studied in the work [ZCM⁺09, ZOT10].

Theorem 1: The problem of answering mCK queries is NP-hard.

Proof:

We prove the theorem by a reduction from the 3-SAT problem. An instance of the 3-SAT consists of $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_n$, where each clause $C_i = \{x_i \vee y_i \vee z_i\}$ ($i = 1, \dots, n$), and $\{x_i, y_i, z_i\} \subset \{u_1, \bar{u}_1, \dots, u_m, \bar{u}_m\}$. The decision problem is to determine whether we can assign a value (true or false) to each variable u_i , $i = 1, \dots, m$, such that ϕ is true. We transform an instance of the 3-SAT problem to an instance of the mCK problem as follows. We consider a circle of diameter d' . Each variable u_i corresponds to a point on the circle, and we place its negation $\bar{u}_i$ diametrically opposite on the circle. Then the distance between u_i and $\bar{u}_i$ is d' . We set $d' = d + \epsilon$, where ϵ is a sufficiently small and positive value, such that the distance between any two points corresponding to different variables is no larger than d .

For each pair of variables u_i and $\bar{u}_i$, we create a keyword q_i ($i = 1, \dots, m$) and associate it with the points corresponding to u_i and $\bar{u}_i$. For each clause C_i , we create a keyword q_{m+i} ($i = 1, \dots, n$) and associate it with the points corresponding to the three variables in C_i . Therefore, given a 3-SAT instance ϕ , we have an mCK query q of $n + m$ keywords. If there exists a result of q of diameter at most d , then there exists a satisfying assignment for $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_n$. On the other hand, a satisfying assignment of the 3-SAT problem determines a set of points with diameter at most d covering all keywords in q . Therefore, the proof is complete. $\square$

3.1.2 Existing Solutions for mCK Queries

bR*-tree Based Method: In the work [ZCM⁺09], Zhang et al. propose a hybrid index structure that combines the R*-tree and bitmap, named bR*-tree. Each R*-tree node is augmented with a bitmap indicating the keywords contained in the objects rooted at this node. Based on the bR*-tree, an exact method is proposed to answer the mCK query. The algorithm adopts an exhaustive search enhanced with several pruning strategies. The search starts from the root node, and is performed in a top-down manner. In each level of the tree, all the candidate combinations of nodes (each combination can cover all query keywords) are generated. For each such combination of nodes, all combinations of

their child nodes that cover all query keywords are generated. This process is repeated until the leaf node level is reached, and then all possible groups are enumerated and the best one is returned as the result.

Virtual bR*-tree Based Method: In the subsequent work [ZOT10], an improved version of the bR*-tree called virtual bR*-tree is proposed. During query processing, the relevant objects and R*-tree nodes are read from the inverted file, and a virtual bR*-tree is built using these relevant objects and nodes in a bottom-up way. The query is then processed using the algorithm proposed in the work [ZCM⁺09] based on the virtual bR*-tree. Compared to the original bR*-tree, the size of the tree is significantly reduced. The experimental results show that this solution is much more efficient than that proposed in the earlier work [ZCM⁺09]. Hence, we use the virtual bR*-tree based method as the baseline, denoted by *VirbR*.

Spatial group keyword query: Cao et al. [CCJO11] study the spatial group keyword (SGK) query. Such a query Q has both a location $Q.\lambda$ and a set of query keywords $Q.\psi$. Given a database of geo-textual objects $\mathcal{O}$, it retrieves a set of objects G , such that $\cup_{o \in G} o.\psi \supseteq Q.\psi$, and the cost of G w.r.t. Q is minimized. The cost function takes into account both the distance of the group to the query and the inter-object distance. That is, the returned group is close to the query location, and the group also has a small diameter. When considering only the diameter of the group and ignoring the query location, such a query is equivalent to the *mCK* query. The pruning methods using location $Q.\lambda$ become invalid in the algorithm proposed for the SGK query, which is reduced to the bR*-tree based algorithm [ZCM⁺09]. Thus it is not compared in our experiments.

Long et al. [LWWF13] propose algorithms for processing the SGK query, but the algorithms cannot handle the case when considering only the inter-object distance, and thus they cannot be used to answer the *mCK* query. They also studied a variant of the SGK query, where the cost function of a group G is defined as $\max_{o_1, o_2 \in \{G \cup Q\}} (\text{Dist}(o_1, o_2))$, called Dia-CoSKQ. That is, the query location is also considered when computing the diameter of the group. They propose both exact and approximation algorithms for this query. We adapt the two algorithms to answer the *mCK* query. Given an *mCK* query q , we select the most infrequent keyword t_{inf} , and on each object o_i containing t_{inf} , we issue a Dia-CoSKQ query with o_i as the query location and $q \setminus o_i.\psi$ as the query keywords, and we invoke the algorithm (either exact or approximate) [LWWF13] to answer the query. After all objects containing t_{inf} are processed, the group with the best cost is used as the result of the *mCK* query. The reason for choosing the most infrequent

keyword t_{inf} is to minimize the number of times for issuing the Dia-CoSKQ query. The two algorithms are denoted by *ASGK* (adapted SGK exact) and *ASGKa* (adapted SGK approximation), respectively. We compare *ASGK* and *ASGKa* with our proposed algorithms in the experiments, and they both have poor performance. The result shows that the adaptation is not suitable for processing the *mCK* query.

3.2 Algorithm *GKG*

We develop a 2-approximation algorithm as a baseline for answering the *mCK* query. We call it the Greedy Keyword Group (*GKG*) algorithm. The algorithm is described as follows. Given a query $q = \{t_{q_1}, t_{q_2}, \dots, t_{q_m}\}$, we first find the most infrequent keyword t_{inf} among the keywords in q based on their frequencies in dataset $\mathcal{O}$. Then, around each object o containing t_{inf} , for each keyword $t \in q \setminus o.\psi$ we find the nearest object containing t . These objects and o form a feasible group, and we denote this group by G_o . After all the objects containing t_{inf} are processed, we select the group that has the smallest diameter to answer the query approximately. We find the most infrequent keyword t_{inf} because this can reduce the number of subsequent operations for finding the nearest object. In the algorithm, we utilize the virtual bR*-tree indexing structure [ZOT10] to find the nearest object containing a term t . The reason for using the virtual bR*-tree is that we use the same index for all methods as the *VirbR* algorithm [ZOT10] for a fair comparison of different algorithms. Alternatively, we can also use other geo-textual indexes (such as IR-tree [CJW09]).

The *GKG* algorithm is detailed in Algorithm 1. We first find the most infrequent query keyword t_{inf} (line 1). For each object o containing t_{inf} , we utilize the virtual bR*-tree indexing structure [ZOT10] to find the nearest object containing a term t to o . Alternatively, we can also use other geo-textual indexes in place (such as IR-tree [CJW09]) and our algorithm *GKG* is equally applicable. We initialize a min priority queue *Queue* to store the nodes and objects traversed, and their minimum distances to o are used as the key. Initially, the root node of the virtual bR*-tree is inserted into *Queue* (line 6). If there are some keywords uncovered, we find the nearest object for each of them (lines 9–20). In each loop, we read the element e that is nearest to o (line 9). If it is an object, we insert it into *group* and remove $e.\psi$ from *ucSet* (lines 9–10). Otherwise, we read each of its child node e' , and insert it into *Queue* only if it covers some keywords in *ucSet* (lines 14–20). We compare the group found around o with the current best group G_{gkg} , and update G_{gkg} if *group* has smaller diameter (line 21).

Algorithm 1: $GKG(q, tree)$

```

1  $G_{gkg} \leftarrow \emptyset$ ;
2  $t_{inf} \leftarrow$  the most infrequent keyword;
3 foreach object  $o$  containing  $t_{inf}$  do
4      $group \leftarrow \{o\}$ ;
5      $Queue \leftarrow$  new min-priority queue;
6      $Queue.Enqueue(tree.root, 0)$ ;
7      $ucSet \leftarrow q.\psi \setminus o.\psi$ ; // uncovered keywords
8     while  $ucSet$  is not empty do
9          $e \leftarrow Queue.Dequeue()$ ;
10        if  $e$  is an object then
11             $group \leftarrow group \cup e$ ;
12             $ucSet \leftarrow ucSet \setminus e.\psi$ ;
13        else
14            foreach entry  $e'$  in node  $e$  do
15                if  $ucSet \cap e'.\psi \neq \emptyset$  then
16                    if  $e$  is a leaf node then
17                         $dist \leftarrow Dist(e', q)$ ;
18                    else
19                         $dist \leftarrow minDist(e', q)$ ;
20                     $Queue.Enqueue(e', dist)$ ;
21        if  $G_{gkg}$  is  $\emptyset$  or  $\delta(group) < \delta(G_{gkg})$  then
22             $G_{gkg} \leftarrow group$ ;
23 return  $G_{gkg}$ ;

```

Approximation ratio and Complexity. We proceed to prove that GKG is within an approximation factor of 2. We denote the group returned by GKG by G_{gkg} , and denote the optimal group for the query by G_{opt} .

Theorem 2: $\delta(G_{gkg}) \leq 2 \cdot \delta(G_{opt})$.

Proof: Let o_i to be the object containing t_{inf} in G_{opt} , and we denote the feasible group containing o_i by G_{o_i} . We know that $\delta(G_{gkg}) \leq \delta(G_{o_i})$ according to the algorithm. We denote by o_f the object that is the furthest to o_i in G_{o_i} .

On one hand, because all objects in G_{o_i} fall in the circle with o_i as center and $Dist(o_i, o_f)$ as radius, $\delta(G_{o_i})$ must be no larger than the diameter of the circle, i.e., $2Dist(o_i, o_f)$. On the other hand, o_f must cover a keyword t_f that is not covered by other objects in G_{o_i} , and it is the nearest object to o_i containing t_f . Hence, in G_{opt} , the object containing t_f must be no closer to o_i than o_f . We thus know that $\delta(G_{opt}) \geq Dist(o_i, o_f)$.

Hence, $\delta(G_{gkg}) \leq \delta(G_{o_i}) \leq 2Dist(o_i, o_f) \leq 2\delta(G_{opt})$. $\square$

In *GKG*, each object containing t_{inf} is considered to be in the candidate group, and we denote the set of such objects as $\mathcal{O}_{t_{inf}}$. For each object $o \in \mathcal{O}_{t_{inf}}$, we find at most $m - 1$ other objects together with o to form a feasible group. If we assume finding the nearest object containing a given keyword costs time d , the time complexity of *GKG* is $O(m|\mathcal{O}_{t_{inf}}|d)$, where d depends on the index structure in place. In our work, the algorithm of finding the nearest object containing a given keyword using the virtual bR^* -tree follows the previous work [ZOT10, HS99], and the complexity is not given in these studies. It is known that it is difficult to give a tight complexity for nearest neighbor query algorithm on R-tree like indexes. The worst case complexity will be $O(|\mathcal{O}'|)$, where $\mathcal{O}'$ indicates the relevant objects are accessed in the keyword query, but in practice it is much better than this.

3.3 SKEC-Based Algorithms

To achieve better accuracy, we propose several novel algorithms with much smaller approximation ratios, which answer the *mCK* query by finding a circle with the smallest diameter that encloses a group of objects covering all query keywords. We call such a circle the “smallest keywords enclosing circle” w.r.t. the given query q , denoted by SKEC_q . We prove that the group enclosed in the circle SKEC_q can answer q with a ratio of $\frac{2}{\sqrt{3}}$, which is very close to 1.

However, finding SKEC_q efficiently remains an open problem, which is challenging because neither its radius nor its center is known, although it is solvable in polynomial-time. We first develop an algorithm for finding SKEC_q exactly and it can answer *mCK* query q with a ratio of $\frac{2}{\sqrt{3}}$. We denote this method by *SKEC*. Algorithm *SKEC* has a high time complexity. To achieve better efficiency, we propose to find SKEC_q approximately, and design two algorithms *SKECa* and *SKECa+*, both of which are able to answer the *mCK* query with a ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$, where ϵ is an arbitrarily small positive value.

We next introduce several definitions and theorems in Section 3.3.1, which lay the foundation of our algorithms. We detail *SKEC*, *SKECa*, and *SKECa+* in Sections 3.3.2, 3.3.3, and 3.3.4, respectively.

3.3.1 Minimum Covering Circle and Keywords Enclosing Circle

Definition 4: Minimum Covering Circle. Given a set of geo-textual objects G , the *Minimum Covering Circle* of G is the circle that encloses them with the smallest diameter, denoted by MCC_G . $\square$

Figure 3.1: Example of two diameters

The problem of finding the minimum covering circle for a given set of objects has been well studied [Meg82, EH72b, EH72a]. Note that the diameter of the circle that encloses a group is different from the diameter of the group. We denote the diameter of a circle $\mathcal{C}$ by $\phi(\mathcal{C})$, and we denote the diameter of a group G by $\delta(G)$. Given a group of objects G , we show that $\delta(G) \neq \phi(\text{MCC}_G)$ in Figure 3.1. Both groups G_1 and G_2 cover keywords $\{t_1, t_2, t_3\}$. In group G_1 , the two diameters are the same. However, in group G_2 , the diameter of G_2 and the diameter of the minimum covering circle of G_2 are different.

Although the diameter of the circle that encloses a group and the diameter of the group can be different, we have the following theorems to show their relationship.

Theorem 3: [EH72b] Given a set of objects G , its smallest object enclosing circle can be determined by at most three points in G which lie on the boundary of the circle. If it is determined by only two points, then the line segment connecting those two points must be a diameter of the circle. If it is determined by three points, then the triangle consisting of those three points is not obtuse. $\square$

Theorem 4: Given a set of objects G and its minimum covering circle MCC_G , we have $\frac{\sqrt{3}}{2}\phi(\text{MCC}_G) \leq \delta(G) \leq \phi(\text{MCC}_G)$.

Proof: 1). If MCC_G is determined by two points in G , according to Theorem 3, the diameter of MCC_G is equal to the distance between the two points, and thus we have $\delta(G) = \phi(\text{MCC}_G)$.

2). Consider that MCC_G is determined by three points A , B , and C in G . We use $\angle A$, $\angle B$, and $\angle C$ to denote the three angles of the triangle consisting of the three points.

First, it is obvious that the distance between any two objects cannot exceed the diameter of the circle, and thus we have $\delta(G) \leq \phi(\text{MCC}_G)$. Second, we assume $\angle C$ is the largest angle. Since $\angle A + \angle B + \angle C = 180^\circ$, we can conclude that $\angle C \geq 60^\circ$. According to Theorem 3, the triangle is not obtuse, and thus we know that $60^\circ \leq \angle C \leq 90^\circ$. Because MCC_G is also the circumcircle of the three points, we obtain that $\frac{\overline{AB}}{\sin \angle C} = \phi(\text{MCC}_G)$ according to the law of sines. Hence, we can get $\sin 60^\circ \leq \frac{\overline{AB}}{\phi(\text{MCC}_G)} \leq \sin 90^\circ$. Because $\delta(G)$ must be no smaller than $\overline{AB}$, we get $\frac{\sqrt{3}}{2}\phi(\text{MCC}_G) \leq \overline{AB} \leq \delta(G)$.

3). When there are more than three points on MCC_G , we can transform this to the cases that MCC_G is determined by either two or three points as follows: first, we select one point on MCC_G , and we remove it from G to check if the minimum covering circle of the remaining points is the same as MCC_G . If so, we ignore it; otherwise, we select the next point and repeat the above process. Finally, we will select two or three points that determines MCC_G , and then we can apply the proof in cases 1) and 2). $\square$

Definition 5: Keywords Enclosing Circle. Given a database of geo-textual objects $\mathcal{O}$ and a set of keywords ψ , the *Keywords Enclosing Circle* w.r.t. ψ (denoted by KEC_ψ) is a circle that encloses a group of objects covering all the given keywords in ψ . We call the one with the smallest diameter the *Smallest Keywords Enclosing Circle* (denoted by SKEC_ψ) $\square$

Figure 3.2: Examples of enclosing circles w.r.t. $\{t_1, t_2, t_3, t_4\}$.

Example 3: As shown in Figure 3.2, given a set of keywords $q = \{t_1, t_2, t_3, t_4\}$, the larger circle is a keyword enclosing circle, and the smaller circle with dashed line is the smallest keyword enclosing circle w.r.t. q . $\square$

From Figure 3.1, it can also be observed that given an mCK query q , the group enclosed by $SKEC_q$ is not necessary the optimal group of q , i.e., G_{opt} . Consider the objects as shown in Figure 3.1, given a query $q = \{t_1, t_2, t_3\}$, G_2 is the optimal group rather than G_1 because $\delta(G_2) < \delta(G_1)$. However, the minimum covering circle of G_1 is $SKEC_q$, because $\phi(MCC_{G_2}) > \phi(MCC_{G_1})$.

Fortunately, we can establish the relationship between the diameter of G_{opt} and the diameter of the group enclosed in $SKEC_q$, denoted by G_{skec} (i.e., $SKEC_q$ is $MCC_{G_{skec}}$), as follows.

Theorem 5: $\delta(G_{skec}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$.

Proof: According to Theorem 4, $\phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$, and $\delta(G_{skec}) \leq \phi(SKEC_q)$. Since $SKEC_q$ has the smallest diameter, we can obtain $\delta(G_{skec}) \leq \phi(SKEC_q) \leq \phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$. $\square$

Theorem 5 shows that if we can find the smallest keywords enclosing circle $SKEC_q$ for a given mCK query q , the group of objects G_{skec} enclosed in $SKEC_q$ can approximately answer query q with a ratio of $2/\sqrt{3}$ (≈ 1.1547). This lays the foundation of our proposed algorithms that find $SKEC_q$ to approximately answer query q , to be presented in the next three subsections.

3.3.2 Algorithm *SKEC*

The *SKEC* algorithm finds the smallest keywords enclosing circle $SKEC_q$ exactly for a given query q . The algorithm has a high time complexity (to be analyzed later). It is based on the following corollary, which follows Theorem 3 and the definition of $SKEC_q$.

Corollary 1: There must exist either three or two objects on the boundary of $SKEC_q$, which determine the circle $SKEC_q$. If two objects determine $SKEC_q$, the line segment connecting them is the diameter of $SKEC_q$.

We denote by $\mathcal{O}'$ the set of objects that contain at least one query keyword. According to the corollary, we can check every combination of two and three objects in $\mathcal{O}'$ to find whether the circle determined by the combination encloses a group covering all query keywords and has the smallest diameter. The circle found finally must be $SKEC_q$. We introduce several pruning strategies to reduce the number of checking in *SKEC*. Before presenting the details of *SKEC*, we introduce the following definition.

Definition 6: Object-across Keywords Enclosing Circle. Given a set of keywords q and an object o , we call a keywords enclosing circle passing through o the *o-across Keywords Enclosing Circle* (denoted by KEC_q^o). The *o-across* keywords enclosing circle with the smallest diameter is called the *o-across Smallest Keywords Enclosing Circle* (denoted by SKEC_q^o). $\square$

Example 4: As shown in Figure 3.2, the larger circle is an o_1 -across keywords enclosing circle, and the smaller circle is the o_1 -across smallest keywords enclosing circle w.r.t. $q = \{t_1, t_2, t_3, t_4\}$. $\square$

Given a query q , there must exist an object containing at least one query keyword on the boundary of SKEC_q . Hence, if we find the object-across smallest keywords enclosing circle (SKEC_q^o) on each object o in $\mathcal{O}'$ (these objects are obtained using the virtual bR*-tree index), the one with the smallest diameter must be SKEC_q . That is, $\text{SKEC}_q = \min_{o \in \mathcal{O}'} \text{SKEC}_q^o$.

The algorithm is shown in Algorithm 2. We first obtain a group G_{gkg} using the GKG algorithm, and the diameter of its minimum covering circle $\text{MCC}_{G_{gkg}}$ serves as the initial upper bound of the diameter of SKEC_q . We denote the current best checked circle by C_{cur} , and C_{cur} is initialized as $\text{MCC}_{G_{gkg}}$ (lines 1–2). For each object o in $\mathcal{O}'$, if it covers all query keywords, we return this object (line 5). Otherwise, we find the smallest o -across keyword enclosing circle, and update C_{cur} if it has a smaller diameter (line 6). Finally, we return the objects in C_{cur} to answer q (lines 7–8).

Algorithm 2: $\text{SKEC}(q)$

```

1  $G_{gkg} \leftarrow \text{GKG}(q)$  ; // invoke GKG
2  $C_{cur} \leftarrow \text{MCC}_{G_{gkg}}$ ;
3  $\mathcal{O}' \leftarrow$  objects containing at least one keyword in  $q$ ;
4 foreach object  $o \in \mathcal{O}'$  do
5   if  $o.\psi = q$  then
6     return  $\{o\}$ ;
7    $\text{findOSKEC}(o, C_{cur})$ ;
8  $G_{skec} \leftarrow$  objects in  $C_{cur}$ ;
9 return  $G_{skec}$ ;

```

Procedure $\text{findOSKEC}()$. We next present how to find SKEC_q^o around each object o in $\mathcal{O}'$. For finding SKEC_q^o , the search space comprises only objects whose distance to o is smaller than $\phi(C_{cur})$. This is because for any object o_f with $\text{Dist}(o, o_f) \geq \phi(C_{cur})$, the

circle passing by o_f and o must have a diameter no smaller than $\text{Dist}(o, o_f)$, and thus is worse than C_{cur} . If these objects together cannot cover all query keywords, o does not need to be processed.

Recall that SKEC_q is determined by either three or two objects in $\mathcal{O}'$. We aim at searching whether the circle determined by object o together with a second object o_j , and the circle determined by o , o_j and a third object o_m is SKEC_q . For each object o_j in the search space of o , we first consider the case when SKEC_q is determined by two objects. We check all the objects in the circle determined by the pair, and update C_{cur} if the circle covers all query keywords. We next consider the case when SKEC_q is determined by three objects. For each object o_m in the search space of o , we check if the circle determined by o , o_j , and o_m can cover all query keywords and has the smallest diameter among all checked circles. The diameter of the circle can be computed using the laws of sines and cosines and we ignore it due to the space limitation.

The second object o_j is processed in ascending order of their distances to o . This assures that when we reach an object o_j with distance to o larger than $\phi(C_{cur})$, we can terminate the search around o immediately, because a circle passing by any further object and o must have a diameter larger than $\phi(C_{cur})$. After o and o_j are fixed, an object o_m is considered as the third object only if $\text{Dist}(o_m, o) < \text{Dist}(o_j, o)$ and $\text{Dist}(o_m, o_j) < \phi(C_{cur})$. The first constraint is because further objects will be processed in subsequent steps, where they are used as the second object. The second constraint is because further objects, together with o and o_j , cannot determine a better circle than C_{cur} .

After all objects in $\mathcal{O}'$ are processed, it is assured that all object-across keywords enclosing circles are found. We use the best one to answer the given $m\text{CK}$ query. The pseudo code is given in Procedure `findOSKEC()`.

First, we read objects whose distances to o are smaller than the diameter of the current best circle C_{cur} (line 1). We process these objects in ascending order of their distances to o . When the second object o_j is fixed, we enumerate the third object o_m to obtain a candidate circle C_{can} (lines 7-10). Next, if C_{can} has a smaller diameter than the current best circle C_{cur} , we check if the objects in C_{can} can cover all query keywords (lines 11-18). If so, we update C_{cur} as C_{can} and begin to enumerate the next circle.

Approximation ratio and Complexity. Algorithm *SKEC* finds SKEC_q exactly and has an approximation ratio of $2/\sqrt{3}$ according to Theorem 5. *SKEC* utilizes the current best circle to prune the search space when finding SKEC_q^o . Assuming that there are n objects in the search space around an object in the worst case, the number of checks in

Procedure findOSKEC(o, C_{cur})

```

1  $olist \leftarrow$  a list of objects in  $\mathcal{O}'$  whose distances to  $o$  are smaller than  $\phi(C_{cur})$ , ranked by
  their distances to  $o$ ;
2 foreach object  $o_j \in olist$  do
3   if  $\text{Dist}(o, o_j) > \phi(C_{cur})$  then
4     break;
5   foreach object  $o_m \in olist$  do
6     if  $\text{Dist}(o_m, o) \geq \text{Dist}(o_j, o)$  or  $\text{Dist}(o_m, o_j) \geq \phi(C_{cur})$  then
7       break;
8     if  $o_j = o_m$  then
9        $C_{can} \leftarrow \text{MCC}_{\{o, o_j\}}$ ;
10    else
11       $C_{can} \leftarrow \text{MCC}_{\{o, o_j, o_m\}}$ ;
12    if  $\phi(C_{can}) \geq \phi(C_{cur})$  then
13       $ucSet \leftarrow q.\psi \setminus o.\psi$  // uncovered keywords
14      foreach object  $o' \in olist$  do
15        if  $o' \in C_{can}$  then
16           $ucSet \leftarrow q.\psi \setminus o'.\psi$ ;
17        if  $ucSet$  is empty then
18           $C_{cur} \leftarrow C_{can}$ ;
19          break;
20 return  $C_{cur}$ ;

```

---

procedure findOSKEC() is $O(n^2)$. We also need to read the objects within a circle, which costs at most $O(n)$. Hence, the time complexity is $O(|\mathcal{O}'|n^3)$. The high time complexity makes *SKEC* impractical especially when n is large (in the worst case $n = |\mathcal{O}'|$).

Considering the high complexity of *SKEC* for finding SKEC_q exactly, we next propose two algorithms for finding SKEC_q approximately with much better efficiency.

3.3.3 Algorithm *SKECa*

In *SKEC*, on each relevant object o we find the o -across smallest keywords enclosing circle (SKEC_q^o) exactly in procedure findOSKEC(). In the *SKECa* algorithm we propose to find SKEC_q^o approximately to gain better efficiency.

3.3.3.1 Finding SKEC_q^o Approximately

The idea of this algorithm is based on the following property of keywords enclosing circles.

Property 1: Given a set of keywords ψ and an object o , if there exists no o -across keywords enclosing circle (KEC_q^o) with diameter D , then no KEC_q^o exists whose diameter is smaller than D .

Proof: This can be proven by contradiction. If there exists an o -across keywords enclosing circle $\mathcal{C}$ with diameter D' smaller than D , at object o we can draw a circumscribed circle $\mathcal{C}_{circ}$ of $\mathcal{C}$ with diameter D , and it is obvious that $\mathcal{C}_{circ}$ is also an o -across keywords enclosing circle, leading to a contradiction.

The property inspires us to use binary search to find the diameter and the position of SKEC_q^o . Given a value D , if we can find a KEC_q^o on object o with diameter D , we know that D must be an upper bound of $\phi(\text{SKEC}_q^o)$, and we will try smaller D in the subsequent search. Otherwise, if using value D we can find no KEC_q^o on object o , D must be a lower bound of $\phi(\text{SKEC}_q^o)$ (Property 1), and we need to enlarge D in the subsequent search. We repeat this process until the gap between the upper bound and the lower bound is smaller than a certain threshold α .

Parameter α is the error tolerance of the binary search, and we present how to set this parameter smartly based on the result of algorithm *GKG* in Section 3.3.3.3. As to be shown in Section 3.3.3.3, by setting a query-dependent value for α based on the result of algorithm *GKG*, we are able to guarantee the approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ for *SKECa*, where ϵ is an arbitrarily small value.

We set the initial upper bound of $\phi(\text{SKEC}_q^o)$ by the diameter of the current best circle, because we aim to find a circle with smaller diameter than the current one. The lower bound can be simply set to 0. However, in order to accelerate the binary search, we use the following lemma to improve the lower bound of $\phi(\text{SKEC}_q^o)$.

Lemma 1: $\phi(\text{SKEC}_q^o) \geq \delta(G_{gkg})/2$, where G_{gkg} is the group found by algorithm *GKG*.

Proof: We denote the group enclosed in SKEC_q^o by G_q . We obtain $\phi(\text{SKEC}_q^o) \geq \delta(G_q)$ (Theorem 4) $\geq \delta(G_{opt}) \geq \delta(G_{gkg})/2$ (Theorem 2). $\square$

As described in Procedure *findAppOSKEC()*, we first set the upper bound of $\phi(\text{SKEC}_q^o)$ by the diameter of the current best circle (line 1). Then we use *searchUB* to search for an o -across keywords enclosing circle by invoking the function *circleScan()* (to be described in Section 3.3.3.2) (line 2). If no such circle exists, we do not need to search on o because $\phi(\text{SKEC}_q^o)$ must be larger than the diameter of the current best circle according to Property 1 (line 3). We set the lower bound of $\phi(\text{SKEC}_q^o)$ according to Lemma 1 (line 4). The binary search stops when the gap between *searchUB* and *searchLB* is smaller than α (lines 5–11).

By replacing the procedure *findOSKEC()* with *findAppOSKEC()* in Algorithm 2, we obtain the *SKECa* algorithm. *SKECa* finds SKEC_q approximately, and the diameter of the circle found is smaller than $\phi(\text{SKEC}_q) + \alpha$.

Procedure findAppOSKEC($o, \delta(G_{gkg}), C_{cur}, \alpha$)

```

1  $searchUB \leftarrow \phi(C_{cur});$ 
2  $oskec \leftarrow \text{circleScan}(o, searchUB);$ 
3 if  $oskec$  is null then
4   | return null
5  $searchLB \leftarrow \delta(G_{gkg})/2;$ 
6 while  $searchUB - searchLB > \alpha$  do
7   |  $diam \leftarrow (searchUB + searchLB)/2;$ 
8   |  $\mathcal{C} \leftarrow \text{circleScan}(o, diam);$ 
9   | if  $\mathcal{C} \neq \emptyset$  then
10  |   |  $searchUB \leftarrow diam;$ 
11  |   |  $oskec \leftarrow \mathcal{C};$ 
12  | else  $searchLB \leftarrow diam$ 
13 return  $oskec;$ 

```

3.3.3.2 Procedure circleScan()

In each step of the binary search in the *SKECa* algorithm, we invoke Procedure *circleScan()* to check if there exists a KEC_q^o with diameter D on a given object o . This procedure is a key operation of Algorithm *SKECa*, and is non-trivial.

The idea of finding a KEC_q^o with diameter D is as follows: If there exists such a circle, besides o there must exist another object falling on the boundary of this circle (we can always rotate the circle around o to assure this). Hence, from the objects whose distance to o is smaller than D , we randomly select one and make it on the boundary of the circle, and we can fix the position of this circle. Then, we rotate this circle from this position, and if at a certain position the objects inside this circle can cover all the query keywords, we know that a KEC_q^o with diameter D is found. If we have rotated the circle back to the beginning position but still cannot find a feasible group, we know that no KEC_q^o with diameter D exists. Figure 3.3 illustrates the sweeping area around an object o given D as the diameter, which is the circle with dashed lines.

We use the virtual bR*-tree to read all objects in the sweeping area. Before the checking on an object o by rotating the circle, we first check whether all keywords can be covered by the objects in the sweeping area, and if not, there exists no KEC_q^o with diameter D and the checking on o is thus avoided.

Now we present how to efficiently rotate the circle to find a KEC_q^o with diameter D . We rotate the circle clockwise. Note that during the rotation, each object in the sweeping area falls on the boundary of the circle only twice. That is, each object is

Figure 3.3: Sweeping area

Figure 3.4: Computing two angles

scanned outside-in or inside-out by the circle exactly once. We map the objects in the sweeping area around o to a polar coordinate system with o as the pole and the horizontal line pointing to the right as the polar axis. On each object o_j in the sweeping area, we compute two polar angles when o_j touches the boundary: 1) when o_j is entering the circle, we find the center point of the circle at the current position and we compute the polar angle of the center point, denoted by $o_j\text{-in}$; and 2) when o_j is exiting the circle, we compute the polar angle of the center point of the circle at the current position, denoted by $o_j\text{-out}$. Figure 3.4 shows an example of computing $o_j\text{-in}$ and $o_j\text{-out}$ in the polar coordinate system with o as the pole.

Next, after we compute the two polar angles for each object in the sweeping area, we sort all these angles in descending order and store them in $\angle$. Initially, we place the circle such that the center point of the circle has the polar angle equal to the largest angle in $\angle$. We check if the objects in this circle can cover all query keywords. If so, a KEC_q^o is found and we return the circle, and otherwise we record the keywords covered by these objects with their frequencies in a table Tab and we start the clockwise rotation from the current position. When the circle is rotated to a position such that the center point of the circle has a polar angle equal to the next largest angle in $\angle$, we update table Tab because the objects enclosed in the circle change. If the angle is an object inside-out angle, we remove from Tab the keywords covered by the object to be rotated out of the circle. If the angle is an object outside-in angle, we update Tab by adding in the keywords covered by the object to be rotated in the circle. If Tab contains all query keywords we terminate the checking, and otherwise we repeat the above rotation process until we find a KEC_q^o or all angles in $\angle$ are reached. The pseudo code is given in Procedure circleScan.

Procedure circleScan($o, diam$)

```

1   $\angle \leftarrow$  empty List;
2  Initialize  $Tab$  by  $q$ ;
3   $G \leftarrow \emptyset$ ;
4  foreach object  $o_j \in \mathcal{O}'$  do
5       $o_j\text{-in} \leftarrow \text{getInAngle}(o, diam, o_j)$ ;
6       $o_j\text{-out} \leftarrow \text{getOutAngle}(o, diam, o_j)$ ;
7       $\angle.\text{addTuple}(o_j\text{-out}, out, o_j)$ ;
8      if  $o_j\text{-out} \neq o_j\text{-in}$  then
9           $G \leftarrow G \cup o_j$ ;                                //  $o_j$  in the initial circle
10          $Tab$  add  $o_j.\psi$ ;
11     else  $\angle.\text{addTuple}(o_j\text{-in}, in, o_j)$ ;
12 if  $Tab$  is full then
13     return  $G$ ;
14 sort  $\angle$  by angle in ascending order;
15 foreach tuple ( $angle, type, o$ ) in  $\angle$  do
16     if  $type = in$  then
17          $G \leftarrow G \cup o$ ;
18         Add  $o.\psi$  to  $Tab$ ;
19         if  $Tab$  is full then
20             return  $G$ ;
21     else
22          $G \leftarrow G \setminus o$ ;
23         Remove  $o.\psi$  from  $Tab$ ;
24 return  $\emptyset$ ;
    
```

Example 5: Figure 3.5 shows an example of the checking process. The outside-in polar angles are marked as blue lines and the inside-out polar angles are marked as red lines. Assume that the rotation reaches angle $o_1\text{-out}$, and before that the keyword-frequency table Tab is $\{t_1:2, t_2:1, t_4:2\}$. Since $o_1\text{-out}$ is an object inside-out angle, we remove the keywords covered by o_1 and we obtain $Tab = \{t_1:2, t_2:1, t_4:1\}$. The rotation stops at the next angle $o_2\text{-out}$, and we update Tab as $\{t_1:1, t_2:1\}$. The next largest angle is $o_4\text{-in}$, and we get $Tab = \{t_1:1, t_2:2, t_3:1\}$. Since Tab still cannot cover all query keywords, we continue the rotation and we reach $o_5\text{-in}$ next. Tab is updated as $\{t_1:1, t_2:2, t_3:1, t_4:1\}$, and a KEC_q^o is found now and we stop the checking and return the result. $\square$

Suppose there are n objects in the sweeping area. The time complexity of sorting the polar angles is $O(n \log n)$. Each outside-in angle and inside-out angle is scanned once during the sweeping. At the initial position the text descriptions of all objects in the sweeping area are read, which has the complexity of $O(n)$, and each subsequent rotation

 Figure 3.5: Circle scan procedure w.r.t. query $\{t_1, t_2, t_3, t_4\}$

only has complexity of $O(1)$. Hence, the rotation and checking together has complexity $O(n)$, and the total time complexity of this procedure is $O(n \log n)$.

3.3.3.3 Approximation ratio and Complexity

SKECa finds a circle under the error tolerance α . Let G_{skeca} denote the group found by *SKECa*. That is, it is guaranteed that $\phi(\text{MCC}_{G_{skeca}}) \leq \phi(\text{SKEC}_q) + \alpha$. We propose an approach to setting a query-dependent threshold value for α such that we can assure an approximate ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ for *SKECa*, where ϵ is an arbitrarily small value. The idea is to utilize the result returned by the 2-approximation algorithm *GKG*. Specifically, after we obtain a group G_{gkg} by invoking *GKG*, we set $\alpha = \epsilon \delta(G_{gkg})/2$, and we have the following lemma.

Theorem 6: The *SKECa* algorithm has an approximation ratio of $\frac{2}{\sqrt{3}} + \epsilon$ by setting $\alpha = \epsilon \delta(G_{gkg})/2$.

Proof: We have $\delta(G_{skeca}) \leq \phi(\text{MCC}_{G_{skeca}})$ according to Theorem 4, and $\phi(\text{SKEC}_q) \leq \frac{2}{\sqrt{3}} \delta(G_{opt})$ according to Theorem 5. Hence, we can obtain $\delta(G_{skeca}) \leq \frac{2}{\sqrt{3}} \delta(G_{opt}) + \alpha = \frac{2}{\sqrt{3}} \delta(G_{opt}) + \epsilon \delta(G_{gkg})/2$. Because $\delta(G_{gkg}) \leq 2\delta(G_{opt})$, we have:

$$\frac{\delta(G_{skeca})}{\delta(G_{opt})} \leq \frac{2}{\sqrt{3}} + \epsilon \frac{\delta(G_{gkg})}{2\delta(G_{opt})} \leq \frac{2}{\sqrt{3}} + \epsilon.$$

Thus we complete the proof. □

The time complexity is also affected by parameter ϵ . The binary search range sr is $\emptyset(\text{MCC}_{G_{kg}}) - \delta(G_{kg})/2 \leq (\frac{2}{\sqrt{3}} - \frac{1}{2})\delta(G_{kg})$. The binary search stops when the gap between the search upper and lower bounds is smaller than α . Hence, the binary search takes $O(\log(sr/\alpha))$ steps, and it can be computed that $sr/\alpha \leq (\frac{2}{\sqrt{3}} - \frac{1}{2})\delta(G_{kg})/(\epsilon\delta(G_{kg})/2) = (\frac{4}{\sqrt{3}} - 1)/\epsilon$.

In the *SKECa* algorithm, in the worst case, the binary search would be performed on all objects in the dataset. Therefore, the time complexity in the worst case is $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$, where $|\mathcal{O}'|$ is the number of objects relevant to the query, $O(\log \frac{1}{\epsilon})$ is the steps of binary search performed on an object, and $O(n \log n)$ is the complexity of one step where n is the number of objects in the sweeping area in the worst case. The `circleScan()` method is invoked only when the sweeping area covers all query keywords on an object. Thus, in practice, the query processing time is much better than analyzed.

3.3.4 Algorithm *SKECa+*

As analyzed in Section 3.3.3.2, the cost of checking whether there exists a KEC_q^o with diameter D is determined by the number of objects in the sweeping area which depends on the value of D . In the *SKECa* algorithm, the binary search is performed on each object, and the minimum diameter of the keywords enclosing circle obtained on processed objects is used as the upper bound in subsequent search on remaining objects. Hence, if on the early processed objects the circles found are large, the upper bound is loose for subsequent search and the checking cost is high.

To avoid this problem and improve efficiency, we propose to perform binary search on all objects in $\mathcal{O}'$ together, instead of on each of them separately. Specifically, we find the diameter and the position of SKEC_q directly using binary search, instead of first finding all object-across smallest keywords enclosing circles and then selecting the best one. We call the process enhanced binary search.

We set the range of the enhanced binary search as follows. Given a query q , if there exists an o -across keywords enclosing circle with diameter D on any object o in $\mathcal{O}'$, D is an upper bound of $\emptyset(\text{SKEC}_q^o)$, and obviously it is also an upper bound of $\emptyset(\text{SKEC}_q)$. If on an object o there exists no o -across keywords enclosing circle with diameter D , D is a lower bound of $\emptyset(\text{SKEC}_q^o)$; if D is a lower bound of $\emptyset(\text{SKEC}_q^o)$ for every o in $\mathcal{O}'$, D is a lower bound of $\emptyset(\text{SKEC}_q)$. The enhanced algorithm is presented in Algorithm 3.

We obtain $\mathcal{O}'$ and set the upper and lower bounds as we do in Algorithm 2. It can be proven that $\delta(G_{kg})/2$ is also a lower bound of SKEC_q by following the proof in Lemma 1.

On each object o , we use array $maxInvalidRange$ to record the largest diameter such that there exists no o -across keywords enclosing circle with diameter $maxInvalidRange[o]$ according to the previous checks. Initially, $maxInvalidRange[o]$ is set to 0 (line 7). The value is used to avoid unnecessary checking on an object o .

Algorithm 3: $SKECa+(q, \alpha)$

```

1   $G_{gkg} \leftarrow GKG(q); C_{cur} \leftarrow MCC_{G_{gkg}};$ 
2   $\mathcal{O}' \leftarrow$  objects containing at least one keyword in  $q$ ;
3   $searchUB \leftarrow \phi(C_{cur});$ 
4   $searchLB \leftarrow \delta(G_{gkg})/2;$ 
5  foreach object  $o$  in  $\mathcal{O}'$  do
6      if  $o.\psi = q$  then
7          return  $\{o\}$ 
8       $maxInvalidRange[o] \leftarrow 0;$ 
9  while  $searchUB - searchLB > \alpha$  do
10      $diam \leftarrow (searchUB + searchLB)/2;$ 
11      $foundResult \leftarrow False;$ 
12     foreach  $o$  in  $\mathcal{O}'$  do
13         if  $diam < maxInvalidRange[o]$  then
14             continue;
15          $oskec \leftarrow circleScan(o, diam);$ 
16         if  $oskec \neq \emptyset$  then
17              $searchUB \leftarrow diam;$ 
18              $C_{cur} \leftarrow oskec;$ 
19              $foundResult \leftarrow True;$ 
20             break;
21         else
22             if  $diam > maxInvalidRange[o]$  then
23                  $maxInvalidRange[o] \leftarrow diam;$ 
24     if  $foundResult = False$  then
25          $searchLB \leftarrow diam;$ 
26  $G_{skeca} \leftarrow$  objects in  $C_{cur};$ 
27 return  $G_{skeca};$ 

```

The enhanced binary search is applied on all objects together (lines 9-24). On each object o in $\mathcal{O}'$, before we find $SKEC_q^o$ with diameter $diam$, we first compare whether $diam$ is less than the invalid diameter stored in $maxInvalidRange$ to avoid unnecessary checking. We can safely discard o if $diam$ is smaller than $maxInvalidRange[o]$ according to Property 1 (lines 12–13). If we can find a keywords enclosing circle KEC_q^o with diameter

$diam$, we update the upper bound of $SKEC_q$ and the current best circle, and we terminate the checking using $diam$ immediately (lines 15–19). Otherwise, we update the maximum invalid diameter of o if it is smaller than $diam$, because no o -across keywords enclosing circle with diameter $diam$ exists (lines 20–22). If on all objects we fail in finding the keywords enclosing circle with diameter $diam$, we increase the lower bound for subsequent search (lines 23–24).

Approximation ratio and Complexity. Since $SKECa+$ finds the same circle as found in $SKECa$, the result group is the same as that returned by $SKECa$, and thus this algorithm also has an approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$.

$SKECa+$ performs binary search on all objects relevant to the query together. The bounds for binary search are the same as those used in $SKECa$, and thus the steps of binary search is also $O(\log \frac{1}{\epsilon})$. $SKECa+$ also invokes the procedure `circleScan()`, which has complexity $O(n \log n)$. In the worst case, the binary search is performed on all objects as well. Therefore, the worst case time complexity of $SKECa+$ is also $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$. However, after we know a keywords enclosing circle with diameter $diam$ exists on an object in $SKECa+$, we can immediately stop the search using $diam$ and avoid the checking on the remaining objects. Therefore, it has much better efficiency in practice than does $SKECa$, which is to be shown in the experimental study in Section 3.5.

The number of objects n in the sweeping area w.r.t. an object depends on the query. If the query keywords are frequent and the optimal result has a large diameter, n would be large. We study the effect of both the frequencies of query keywords and the diameter of the optimal result on the efficiency of our algorithms in the experiments.

3.4 Exact Algorithm

It is challenging to develop an efficient exact algorithm for mCK queries, as an exact algorithm cannot avoid an exhaustive search in the object space. The best known solution [ZOT10] performs exhaustive search on all objects that contain at least one query keyword, and thus the computational cost is high if such objects are large in number. In this section, we propose an exact algorithm, which leverages the result of our approximation algorithm $SKECa+$ to greatly reduce the space of exhaustive search.

3.4.1 Algorithm Framework

If we know the smallest circle that encloses the optimal group G_{opt} , i.e., $MCC_{G_{opt}}$, we can do exhaustive search only on objects in $MCC_{G_{opt}}$ and the search space can be reduced significantly, compared to the search space of the best known algorithm [ZOT10]. However, we know neither the radius nor the center of $MCC_{G_{opt}}$. Note that the circle $MCC_{G_{opt}}$ is different from the smallest keywords enclosing circle w.r.t. q , i.e., $SKEC_q$. For example, as shown in Figure 3.1, given a query $q = \{t_1, t_2, t_3\}$, $SKEC_q$ is MCC_{G_1} , but the optimal group G_2 is enclosed in MCC_{G_2} .

Although we do not know the exact size of $MCC_{G_{opt}}$, we prove that the diameter of $MCC_{G_{opt}}$ can be bound by the diameter of $SKEC_q$ as follows.

Lemma 2: $\phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}} \phi(SKEC_q)$.

Proof: Denote the group enclosed in $SKEC_q$ by G_{skec} . According to Theorem 4, $\phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}} \delta(G_{opt})$, and $\phi(SKEC_q) \geq \delta(G_{skec})$. Because G_{opt} has the smallest diameter, we have $\phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}} \delta(G_{opt}) \leq \frac{2}{\sqrt{3}} \delta(G_{skec}) \leq \frac{2}{\sqrt{3}} \phi(SKEC_q)$. $\square$

With Lemma 2, we propose to utilize $SKEC_q$ to reduce the exhaustive search space. Our basic idea is that, we find the circles that might be $MCC_{G_{opt}}$ utilizing $SKEC_q$. Within each such candidate circle, we perform an exhaustive search (with some pruning strategies) on the objects enclosed by it to find a group. After all candidate circles are found and checked, the best group found must be the optimal group. Note that we use *SKECa+* to find approximate $SKEC_q$, rather than *SKEC* to find the exact $SKEC_q$, which is computationally expensive. The circle returned by *SKECa+* has a diameter not exceeding $\phi(SKEC_q) + \alpha$. Thus, $\phi(MCC_{G_{opt}})$ is bound by $\frac{2}{\sqrt{3}}$ times the diameter of the circle returned by *SKECa+* as well.

We proceed to explain how to find the candidate circles that might be $MCC_{G_{opt}}$. Since $MCC_{G_{opt}}$ is the smallest circle that encloses the optimal group G_{opt} , there must exist an object in G_{opt} on the boundary of $MCC_{G_{opt}}$. The problem is that we do not know this object, which can be any object o containing at least one query keyword. To avoid finding candidate circles on each such object o , we establish the following lemma to filter out objects that cannot be on the boundary of $MCC_{G_{opt}}$. We denote by G_{skeca} the group found by *SKECa+* and the smallest circle enclosing G_{skeca} by $MCC_{G_{skeca}}$.

Lemma 3: On an object o , if the diameter of $SKEC_q^o$ is larger than $\frac{2}{\sqrt{3}}$ times the diameter of $MCC_{G_{skeca}}$, o cannot be on the boundary of $MCC_{G_{opt}}$.

Proof: If o is on the boundary of $MCC_{G_{opt}}$, $\phi(SKEC_q^o) \leq \phi(MCC_{G_{opt}})$ (note that even though an object o is indeed the object on the boundary of $MCC_{G_{opt}}$, $SKEC_q^o$ may still not be $MCC_{G_{opt}}$). Because $\phi(MCC_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\phi(SKEC_q)$ (according to Lemma 2), and $\phi(MCC_{G_{skeca}}) \geq \phi(SKEC_q)$ (because of the binary search), we have $\phi(SKEC_q^o) \leq \frac{2}{\sqrt{3}}\phi(MCC_{G_{skeca}})$, leading to a contradiction. $\square$

With Lemma 3, we prune some objects from consideration for finding candidate circles as follows: Recall that in *SKECa+*, we use an array *maxInvalidRange* to store the maximum diameter on each object o such that there exists no KEC_q^o with diameter *maxInvalidRange*[o]. This means that $\phi(SKEC_q^o)$ must be larger than *maxInvalidRange*[o]. Hence, in the exact algorithm, we can discard o if *maxInvalidRange*[o] $\geq \frac{2}{\sqrt{3}}\phi(MCC_{G_{skeca}})$ according to Lemma 3. If an object o cannot be pruned, o is possibly on the boundary of $MCC_{G_{opt}}$, and we find all candidate circles that cover all query keywords and pass through o , and in each of them we perform an exhaustive search.

Algorithm 4: *EXACT* (q, α)

```

1  $G_{skeca} \leftarrow SKECa+(q, \alpha);$ 
2  $diam \leftarrow \frac{2}{\sqrt{3}}\phi(MCC_{G_{skeca}});$ 
3  $bestGroup \leftarrow G_{skeca};$ 
4 get the array maxInvalidRange computed in SKECa+;
5 foreach  $o$  in  $\mathcal{O}'$  do
6   | if maxInvalidRange[ $o$ ] < diam then
7   |   |  $circleScanSearch(o, diam, bestGroup);$ 
8 return bestGroup;
```

The pseudocode is described in Algorithm 3. We first invoke *SKECa+* to get an approximate result (line 1). Then we compute the upper bound *diam* of the diameter of the smallest circle that encloses the optimal group (line 2). On each object o , if it cannot be pruned (line 6), we invoke Procedure *circleScanSearch*() to find all keywords enclosing circles with diameter *diam* around o , to do exhaustive search in each such circle, and to update the best group (line 7). After all objects in $\mathcal{O}'$ are processed, we return *bestGroup* as the result.

Procedure *circleScanSearch*(). In this procedure, we first find all o -across keywords enclosing circles with diameter $\frac{2}{\sqrt{3}}\phi(MCC_{G_{skeca}})$ on an object o , which are candidate circles. The idea is similar to that of Procedure *circleScan*(). We first fix a sweeping area around o that contains all objects relevant to the query whose distances to o are smaller than $\frac{2}{\sqrt{3}}\phi(MCC_{G_{skeca}})$. In this sweeping area, we rotate the circle around o with diameter

$\frac{2}{\sqrt{3}} \phi(\text{MCC}_{G_{skcca}})$ clockwise. We also use a table *Tab* to store all keywords with their frequencies covered in the rotating circle, and update it once an object is rotated inside-out or outside-in. Once all query keywords are contained in *Tab*, we know that a KEC_q^o (which is a candidate circle) is found.

In $\text{circleScan}()$, once a KEC_q^o is found it returns immediately, because the procedure only checks if there exists a KEC_q^o with a given diameter. In $\text{circleScanSearch}()$ we need to perform an exhaustive search to find the best group in the circle. Hence, after the group is found, we still need to repeat the above process to find all candidate circles on o , and to enumerate the best group in each of them. We invoke $\text{Procedure search}()$ to perform exhaustive search to enumerate the group with the smallest diameter in a candidate circle, which is presented as follows.

3.4.2 Procedure search()

The $\text{search}()$ procedure is designed as a branch-and-bound process. We adopt the depth-first-search strategy to do enumeration. We use *selectedSet* to store the objects that are already selected in the current enumeration. The object o on which the search is performed is always in *selectedSet* since it must be contained in the generated group. We use *candidateSet* to store the objects in the circle that are possible to combine with objects in *selectedSet* to form a group whose diameter is smaller than that of the current best group.

In each step, we select one object from *candidateSet* and check if combining it with the objects in *selectedSet* can generate a better group. If so, we remove this object from *candidateSet* and add it to *selectedSet*. This step is performed iteratively. The level of this depth-first-search is at most the number of query keywords, because each enumerated object contains at least one new query keyword. The current best group *curGroup* is updated when a group covering all the query keywords with smaller diameter is found.

The search complexity is exponential with the number of objects relevant to the query in a candidate circle. We develop several pruning strategies utilizing both textual and spatial properties in $\text{search}()$ to improve efficiency.

Pruning Strategy 1. Given an object o , and let G denote the group $\text{selectedSet} \cup o$, if the diameter of G exceeds the diameter of the current best group *curGroup*, o does not need to be added to *selectedSet*. This is because that, for any group G' generated from G , it is true that $\delta(G') \geq \delta(G)$. Hence, the diameter of G' is also larger than $\delta(\text{curGroup})$, which means that no better groups can be obtained from G .

Pruning Strategy 2. If an object o cannot contribute any new keyword to *selectedSet*, o is not necessary to be added to *selectedSet*. This can be justified as follows: Denote the group $\text{selectedSet} \cup o$ by G . *selectedSet* and G cover the same set of query keywords and $\delta(\text{selectedSet}) \leq \delta(G)$, and thus G can be pruned.

Pruning Strategy 3. If the objects in *candidateSet* cannot cover the keywords that have not be covered by *selectedSet*, we can stop the search using *selectedSet*. This is because that even if we select all the objects in *candidateSet*, and combine them with the objects in *selectedSet*, a group covering all the query keywords cannot be generated.

Procedure $\text{search}(q, \text{selectedSet}, \text{candidateSet}, \text{maxId})$

```

1  if  $\text{selectedSet}.\psi = q$  then
2    if  $\delta(\text{selectedSet}) \leq \delta(\text{curGroup})$  then
3       $\text{curGroup} \leftarrow \text{selectedSet}$ ;
4    return  $\text{curGroup}$ ;
5  if  $\delta(\text{selectedSet}) > \delta(\text{curGroup})$  then
6    return  $\emptyset$ ;
7   $\text{nextSet} \leftarrow \emptyset$ ;
8   $\text{leftKeywords} \leftarrow \emptyset$ ;
9  foreach candidate object  $o_c$  in candidateSet do
10    if  $\delta(\text{selectedSet} \cup o_c) > \delta(\text{curGroup})$  then
11      continue
12    if  $(q - \text{selectedSet}.\psi) \cap o_c.\psi = \emptyset$  then
13      continue
14    if  $o_c.Id < \text{maxId}$  then
15      continue
16     $\text{nextSet} \leftarrow \text{nextSet} \cup o_c$ ;
17     $\text{leftKeywords} \leftarrow \text{leftKeywords} \cup o_c.\psi$ ;
18 if  $\text{leftKeywords} \cup \text{selectedSet}.\psi \neq q$  then
19   return  $\text{curGroup}$ ;
20 foreach object  $o_n$  in nextSet do
21    $\text{newSelSet} \leftarrow \text{selectedSet} \cup o_n$ ;
22    $\text{newCandSet} \leftarrow \text{nextSet} \setminus o_n$ ;
23    $\text{group} \leftarrow \text{search}(q, \text{newSelSet}, \text{newCandSet}, o_n.Id)$ ;
24   if  $\delta(\text{group}) < \delta(\text{curGroup})$  then
25      $\text{curGroup} \leftarrow \text{group}$ ;
26 return  $\text{curGroup}$ ;

```

In Procedure $\text{search}()$, if *selectedSet* already covers all query keywords, we compare this group with the current best group, and we return the better one as the result (lines 1–4). We then check if the selected group is already worse than the current best solution (line 5). In lines 8–13, we scan *candidateSet* and filter out objects that cannot

be combined with objects in *selectedSet*. Line 9 applies the pruning strategy 1, line 10 applies the pruning strategy 2, and line 11 is used to avoid enumerating duplicate groups. We check whether the pruning strategy 3 is satisfied in lines 14–15. After we add a new object to *selectedSet*, we invoke the procedure recursively to find the group with newly selected object set *newselSet* and new candidate object set *newcandSet* and update *curGroup* correspondingly (lines 17–21).

Complexity. If there are relevant n objects in the sweeping area in `circleScanSearch()` in the worst case, the complexity of computing and sorting the rotation angles is $O(n \log n)$. The exhaustive search in the sweeping area has complexity $O(n^{|q|-1})$, because the depth-first search level is at most $|q|$ and the object on which the search is performed already contains at least one query keyword. Since the diameter of the candidate circle is bounded as shown in Lemma 2, n is usually not large. In summary, on an object the total search complexity is $O(n \log n + n^{|q|-1}) \approx O(n^{|q|-1})$. If there are $\mathcal{O}'$ objects relevant to the query, the worst case time complexity of *EXACT* is $O(|\mathcal{O}'|n^{|q|-1})$, where in general $n \ll |\mathcal{O}'|$. In practice, we only do the search on objects satisfying the bound as described in Lemma 3, and thus the practical performance is better than analyzed. Note that the worst case time complexity of the best known solution [ZOT10] is $O(|\mathcal{O}'|^{|q|})$, which is worse than *EXACT*. As to be shown in the experimental study in Section 3.5, *EXACT* is much more efficient.

3.5 Experiments

3.5.1 Experimental Settings

Algorithms. We evaluate four approximation algorithms, namely *GKG* (in Section 3.2), *SKEC* (in Section 3.3.2), *SKECa* (in Section 3.3.3), and *SKECa+* (in Section 3.3.4), and the exact algorithm *EXACT* (in Section 3.4). We also compare our exact algorithm with the state-of-the-art solution proposed in the work [ZOT10], denoted by *VirbR*, and the methods of adapting the spatial group keyword query [LWWF13], denoted by *ASGK* and *ASGKa* (Section 3.1.2).

Datasets. We use three real-life datasets. Table 3.1 lists some properties of these datasets. Datasets NY and LA are crawled using Google Place API in New York and Los Angeles, respectively. Each crawled object has a name and a type such as “food” and “restaurant,” used as the textual description of the object, and a pair of latitude and longitude representing its location. Dataset TW is crawled from Twitter within the

Dataset	Number of Objects	Unique words	Total words
NY	485,059	116,546	1,143,013
LA	724,952	161,489	1,833,486
TW	1,000,100	487,552	5,170,495

Table 3.1: Dataset properties

area of USA. Each geo-tweet is treated as a geo-textual object, and its content is used as the description of the object and its latitude and longitude are used as the geo-location of the object.

The location of an object is in form of a pair of latitude and longitude. In order to compute the Euclidean distance between locations, we convert the data to the UTM (Universal Transverse Mercator coordinate system) format, using World Geodetic System 84 specification.

Query generation. We generate 5 query sets with different numbers of keywords, i.e., 2, 4, 6, 8, and 10, for each dataset. Each set comprises 50 queries. According to the complexity analysis of the proposed approximation and exact algorithms, their runtimes are affected by the diameter of the optimal group w.r.t. a given query. When evaluating the effect of a certain parameter, we try to bound the diameter of the optimal group for a query, so that the effect of the diameter does not vary too much for queries in one set. For example, to set the upper bound diameter at 20% of the diameter of the whole dataset, we first randomly draw a circle with diameter no larger than 20% of the diameter of all objects in the dataset, and then we randomly select the terms that appear in this circle according to their frequencies. This makes sure that the diameter of the optimal group cannot exceed 20% of the diameter of the whole dataset. It is hard to impose a lower bound constraint when generating queries. We also study the effect of the diameter bound on the efficiency of algorithms.

We set the number of query keywords to 6 and the diameter bound to 20% of the diameter of a dataset by default in our experiments.

Setup. The virtual bR*-tree index structure is disk resident, and the page size is set to 4KB. The number of children of a node in the tree is set to 100. All algorithms were implemented in C++ and run on Linux with a 2.66GHz CPU and 8GB RAM.

3.5.2 Experimental Results

3.5.2.1 Tuning the binary search parameter ϵ

The value of ϵ affects both efficiency and accuracy of *SKECa* and *SKECa+*. We vary ϵ from 0.0004 to 0.25. Figures 3.6(a) and 3.6(b) show the runtime and the accuracy of

SKECa and *SKECa+* when we vary ϵ on the LA dataset, respectively.

Figure 3.6: Varying ϵ

It can be observed that *SKECa* always runs slower than *SKECa+*. Recall that the steps of binary search needed to be performed is linear with $O(\log \frac{1}{\epsilon})$ in *SKECa*. Hence, as ϵ increases, *SKECa* runs faster because a larger ϵ reduces the steps of binary search performed on each object. *SKECa* and *SKECa+* return the same result, and the difference is the way of performing the binary search. Parameter ϵ also determines the steps of binary search in *SKECa+*. However, since *SKECa+* performs the binary search on all objects relevant to the query together, its runtime is only slightly affected and is much better than that of *SKECa*.

The two algorithms have the same accuracy, and the accuracy drops as ϵ becomes larger. Using a smaller ϵ can obtain a circle that is more close to SKEC_q , and thus the group enclosed in the circle found finally is more close to the optimal result. This is consistent with the ratio of the two algorithms, i.e., $(\frac{2}{\sqrt{3}} + \epsilon)$.

Because *SKECa+* outperforms *SKECa* consistently, we report the results of *SKECa+* in subsequent experiments only. Based on Figure 3.5, we set ϵ to 0.01 for *SKECa+* by default, because it strikes a good tradeoff between accuracy and efficiency. Similar results are observed on the other two datasets, and we report them in Appendix 3.5.2.5.

3.5.2.2 Varying the number of query keywords

Figure 3.7 shows the runtime (in logarithmic scale) and accuracy of the five algorithms, i.e., *GKG*, *SKECa+*, *EXACT*, *VirbR*, and *ASGK* when we vary the number of query keywords on each dataset.

On all datasets, *SKECa+* achieves better accuracy than does *GKG*, because it has a better approximation ratio. *SKECa+* can always obtain nearly optimal groups. *GKG* runs the fastest on all datasets. It is interesting to observe that *GKG* runs faster as the number of query keywords increases on some datasets. The reason is as follows. *GKG* only searches for a group of objects that contain the most infrequent query keyword. Given more query keywords, the most infrequent keyword is more likely to have lower frequency, and thus fewer objects need to be checked. *SKECa+* runs slower as the number of query keywords increases. Recall that the complexity of *SKECa+* is $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$. Given more query keywords, the number of objects relevant to the query $|\mathcal{O}'|$ becomes larger. In addition, the number of relevant objects in the sweeping area n in Procedure `circleScan()` also increases.

EXACT outperforms *VirbR* by more than an order of magnitude on all datasets when queries contain more than 4 keywords. *EXACT* can answer most queries within 10 seconds, but *VirbR* takes several minutes to answer a query in average when queries contain 8 or 10 keywords. Recall that the complexity of *EXACT* is $O(|\mathcal{O}'|n^{|q|-1})$. As the number of query keywords increases, the number of objects relevant to the query $|\mathcal{O}'|$ increases, and the number of objects in the sweeping area n also increases, and hence *EXACT* runs slower. The complexity of *VirbR* is $O(|\mathcal{O}'|^{|q|})$, and thus it also runs slower as the number of query keywords increases. Recall that the *EXACT* algorithm first invokes *SKECa+* to reduce the search space then it performs the exhaustive search. The experimental results show that *SKECa+* is able to prune the search space significantly, thus making *EXACT* efficient.

We observe that the performance of *ASGK* is much worse than *VirbR* and *EXACT*. Both the efficiency and accuracy of *ASGKa* are much worse than *SKECa+*. They are not originally developed to process the *mCK* query, and the results show the adaption of the algorithms [LWWF13] for processing the *mCK* query is not efficient. We will ignore *ASGK* and *ASGKa* in the rest of the experiments.

Figure 3.8 shows the comparison of the algorithms *SKEC* and *SKECa+* on dataset LA. The runtime of *SKECa+* is much better than *SKEC*, which is consistent with the analysis of the time complexity of the two algorithms in Section 3.3. *SKEC* is extremely slow when the number of query keywords is large, because the number of relevant objects ($\mathcal{O}'$) becomes larger, and the worst case complexity of this algorithm is $O(|\mathcal{O}'|^4)$. When there are more than 6 keywords, lots of queries take at least 5 minutes to finish, and thus we only report the results on query sets containing 2, 4, and 6 keywords to make

Figure 3.7: Varying number of query keywords

the figure readable. It can be observed that the two algorithms have similar accuracy,

because we set ϵ to 0.01, a very small value to control the error of *SKECa+*. The relative performance comparisons between *SKEC* and *SKECa+* are qualitatively similar in other experiments and we do not report them.

Figure 3.8: Comparing *SKEC* with *SKECa+*

3.5.2.3 Varying the optimal group diameter bound

In this set of experiments, we study the effect of the diameter bound of the optimal group for a query. We vary the diameter bound from 10% to 30% of the diameter of a dataset. Figures 3.9 and 3.10 show the results on LA and TW.

Figures 3.9(a) and 3.10(a) show the runtime of two approximation algorithms on LA and TW. *GKG* searches for a group around each object containing the most infrequent keyword, and the group diameter does not affect the runtime of *GKG*. As the diameter bound increases *SKECa+* runs slower. The reason is that with a larger bound the optimal group for a query is more likely to have a larger diameter, and *SKECa+* needs to scan a larger sweeping area to find a keywords enclosing circle with the given diameter. From Figures 3.9(b) and 3.10(b) we can see that *SKECa+* has better accuracy than *GKG* and is always able to achieve nearly optimal results.

Due to the hardness of answering *mCK* queries, the exact algorithms may be very slow on some queries which dominate the average running time. For the readability of our figures, we set a timeout threshold to 1 minute. We observe that the queries where *VirbR* succeeds to find the result within the timeout threshold can always be answered by our algorithm *EXACT*. When comparing the two algorithms, we only report on queries where both algorithms succeed to return a result within the time limit.

Figures 3.9(c) and 3.10(c) show the runtime of *EXACT* and *VirbR* (y -axis in logarithmic scale) on LA and TW. It is shown that *EXACT* outperforms *VirbR* by about one order of magnitude on those queries that both algorithms can finish in 1 minute. The success rate of two algorithms is shown in Figures 3.9(d) and 3.10(d). *EXACT* always has a better success rate (close to 100%). Both algorithms have lower success rate as the diameter bound increases. This is because that if the optimal group has a larger diameter the exhaustive search takes longer time in both algorithms. We observe similar result on NY and ignore it.

Figure 3.9: Varying optimal group diameter bound on LA

To further study these slow queries, Figure 3.11 compares the runtime and success rate of *EXACT* and *VirbR*, when the timeout threshold varies from 15 seconds to 4

Figure 3.10: Varying optimal group diameter bound on TW

minutes and the diameter is bounded by 30% of the whole space. *EXACT* solves most queries within 15 seconds and it always outperforms *VirbR*.

We study the cases where *EXACT* greatly outperforms *VirbR*, and we find that the result groups of these queries have small diameters. Recall that *EXACT* first reduces the search space by utilizing *SKECa+* and then performs the exhaustive search in the reduced space. For queries that cannot be solved within 15 seconds by both algorithms, we find that these queries contain keywords with both high and low frequency and the diameter of the result group is large.

Figure 3.11: Varying timeout threshold

3.5.2.4 Varying the query keywords frequencies

In this set of experiments, we vary the frequencies of query keywords and evaluate the performance of five algorithms on LA, i.e., *GKG*, *SKECa+*, *EXACT*, and *VirbR*. We rank terms in ascending order of their frequencies, and then generate a query set using lower $x\%$ terms, i.e., we select terms from the $x\%$ least frequent terms to form a query. We vary $x\%$ from 20% to 100% to generate 5 query sets (100% means that the query keywords are selected from all terms in a dataset according to their frequency, as we do in previous experiments).

Figure 3.12 shows the runtime and accuracy of four algorithms. It can be observed in Figure 3.12(a) that as the frequency of query keywords increases, both approximation algorithms run slower. This is because that more objects need to be taken into consideration during algorithm execution. The runtime of *EXACT* and *VirbR* is reported only on queries that can be answered within the 1 minute threshold. *EXACT* has better success rate as shown in Figures 3.12(d); on the queries that both algorithms succeed, *EXACT* is almost one order of magnitude faster than *VirbR*, as shown in Figure 3.12(c) (y -axis in logarithmic scale). *EXACT* and *VirbR* run slower as query keywords become more frequent, which is consistent as analyzed. We observe similar results on the other two datasets and they are not reported.

3.5.2.5 Tuning the binary search parameter ϵ

We also study the effect of ϵ on the efficiency and accuracy of *SKECa* and *SKECa+* on datasets NY and TW. We vary ϵ from 0.0004 to 0.25. Figures 3.13(a) and 3.13(b)

Figure 3.12: Varying query keywords frequencies

show the runtime and the accuracy of *SKECa* and *SKECa+* when we vary ϵ on the NY dataset, respectively. Figures 3.13(c) and 3.13(d) show the runtime and the accuracy of *SKECa* and *SKECa+* when we vary ϵ on the TW dataset, respectively.

It can be observed that the similar results are obtained to that on LA. *SKECa* always runs slower than *SKECa+*. As ϵ increases, their runtime drops, but the accuracy becomes worse. We observe that on NY and TW, setting ϵ to 0.01 can also balance the efficiency and accuracy well. Thus, we use 0.01 as the default value of ϵ for all experiments on all datasets.

(a) Runtime (NY)

(b) Appr. Ratio (NY)

(c) Runtime (TW)

(d) Appr. Ratio (TW)

Figure 3.13: Varying ϵ

3.5.2.6 Scalability

To evaluate scalability, we use 5 datasets containing tweets with locations, all of which are crawled from Twitter. The largest dataset contains 5 million tweets, and we sample other datasets from it. Figure 3.14 shows the runtime and approximation ratio of four algorithms on TW, i.e., *GKG*, *SKECa+*, *EXACT* and *VirbR* (the number of query keywords is 6). Both approximation algorithms scale quite well with the size of the dataset, and all queries can be answered within 1 second by *GKG* and *SKECa+*. The *EXACT* algorithm also scales well. *VirbR* runs slower than *EXACT* by orders of magnitude, and it takes more than one minute to answer a query in average when the dataset contains

more than 3 million objects. The accuracy changes only slightly, and *SKECa+* always returns nearly optimal results and has better accuracy than *GKG*.

Figure 3.14: Scalability

3.6 Summary

We study the problem of answering *mCK* queries in this work. We prove that this problem is NP-hard, which is not established in previous work. We propose a 2-approximation greedy approach as a baseline. Utilizing this greedy method, we first devise an approximation algorithm *SKEC* that aims at finding the smallest circle that can enclose a group of objects covering all query keywords. We prove that its approximation ratio is $\frac{2}{\sqrt{3}}$. *SKEC* has a high complexity, and we design another two approximation algorithms, *SKECa* and *SKECa+*, to find such a circle approximately for better efficiency. Their approximation ratio is $(\frac{2}{\sqrt{3}} + \epsilon)$, where ϵ can be an arbitrarily small positive value. We also design an exact algorithm utilizing *SKECa+* to reduce the exhaustive search space significantly. Extensive experiments were conducted, which verifies our theoretical analysis and shows that our exact algorithm outperforms the best known solution by an order of magnitude. In the future, it would be of interest to investigate the problem of answering the *mCK* query in a distributed setting.

Chapter 4

Representative Geospatial Data Exploration

This chapter is organized as follows: Section 4.1 formally defines the SOS and ISOS problems. In Section 4.2 we introduce our proposed greedy algorithm for SOS problem and show the approximation ratio. In Section 4.3, we extend the greedy algorithm for ISOS problems with pre-fetching strategy. A sampling method is introduced in Section 4.4 to accelerate the proposed algorithm while controlling the result in desired error tolerance. Section 4.5 presents the experiments of all our proposed algorithms. Finally we summarize our contributions of this chapter in Section 4.6.

4.1 Problem Statement

We first introduce two concepts and the Spatial Object Selection (SOS) problem, which is to support end-users to explore a geospatial dataset on the limited space of a map. Then we introduce the Interactive Spatial Object Selection (ISOS) problem.

4.1.1 Representative and Visibility Constraints

A geospatial object o is represented by a triple $o = \langle \lambda, \omega, \mathcal{A} \rangle$, where $o.\lambda$ is the location where o is posted, $o.\omega$ is the weight (normalized in $[0, 1]$), which can be either computed from some attributes to represent the popularity or importance of the object, or simply be assigned with a unit weight, and $o.\mathcal{A}$ is a set of attributes of the object. For example, a geo-tagged tweet is a geospatial object. The textual content of the tweet is an attribute of the object. A geo-tagged image is also a geospatial object. The image content is

viewed as an attribute of the geospatial object. In this thesis, we consider a collection of geospatial objects, denoted by O .

It is overwhelming and annoying to display all geospatial objects to end users in a window of a map. It remains a challenge on how to select a small set S of geospatial objects to represent the collection O of geospatial objects, where $S \subseteq O$.

Representative Constraint Usually, a larger S leads to a more representative set. However, visualizing too many geospatial objects in a window or screen of a map is overwhelming for users to find out truly useful information. For example in Google Maps, around 500 geospatial objects are displayed to users in a single window [DSL⁺12]. Therefore, we aim to enforce a *representative constraint*: choose k objects to represent the set of geospatial objects falling within the region of user interest, where different users can have different specifications of k . We assume w.l.g. that k is much smaller than the set of objects falling within the region of interest.

We compute the representative score of an object by its similarity with other objects. We denote the similarity between two objects o_i, o_j by a function $Sim(o_i, o_j)$ that is computed from the attributes $o_i.\mathcal{A}$ and $o_j.\mathcal{A}$, and then normalized in $[0, 1]$. In this thesis, we leave $Sim(., .)$ as a general function. We believe that a general function is important for us to cope with various types of resources to meet the needs in different scenarios. For example, we may use textual similarity and geospatial distance to measure the similarity of two geo-tagged tweets.

With a given similarity function $Sim(o_i, o_j)$ between objects, we define the similarity between an object o and a set S of objects by:

$$Sim(o, S) = \max_{o' \in S} Sim(o, o') \quad (4.1)$$

Equation 4.1 measures how well an object o can be represented by set S . Intuitively, each object o is represented by the object in S that is most similar to o . Our proposed solution can also be extended to handle other aggregation metrics, such as *sum* or *avg*. For the ease of presentation, we only discuss *max* in this thesis.

We next define the representative score of S , namely $Score(S)$, by the similarity between S and O , which incorporates the weight of each object o and is evaluated by

$$Score(S) = Sim(O, S) = \frac{1}{|O|} \sum_{o \in O} o.\omega \times Sim(o, S) \quad (4.2)$$

Here we combine the weight of o and the similarity between S and o , which can be viewed as the utility of object o . Equation 4.2 aims to maximize the total utilities of all the objects, where similar definition is also used in [MSV15].

Visibility Constraint. Similar to the previous work on query result diversification and cartographic selection (e.g. [DP12, DSLG⁺12, NAS12]), we also enforce that any two selected objects should not be too close to each other, so that users can distinguish them on the map.

4.1.2 Advantages of the Representative Score

Maximizing the representativeness of all the objects. To describe how well a subset S can represent the whole collection of geospatial objects O , an intuitive idea is to define how well S represents each single object o in O (Equation 4.1). Since the representativeness of S on an object o_1 should be irrelevant to that of S on another object o_2 , the representative score of S w.r.t. O is simply an aggregation of the representative score over each object in O (Equation 4.2). Then our objective is to find S that can maximize the $Sim(O, S)$.

Maximizing the utility of each object. Instead of treating each object equally, we combine the weight of each object o and the representativeness of S on it as the utility of o , such that the important objects are more likely to be represented. Intuitively, if an object o can be represented by other objects in S , the similarity between o and the objects in S should be high. In particular, an object should always represent itself. That explains why we have the weight associated with objects in S in Equation 4.2.

Supporting various kinds of geospatial objects. Since we use the attributes $o.A$ of the object to measure the similarity, different types of geospatial objects can be supported by our method. Another salient feature is that for a specific type of objects, we can vary the definition of the similarity function for different applications without ad-hoc algorithms.

Extending map exploration. The efficient exploration feature introduced in Figure 1.2(c) can be directly supported by the definition of **Representative Constraint**. Each object o that is not shown in the map ($o \in O - S$) is represented by a selected object o' ($o' \in S$), $Sim(o, S) = Sim(o, o')$. It indicates that o' is the most similar object to o among all objects shown in the window, and if we view o' for details o can be displayed as an extension of map exploration.

A user study in Section 4.5.2 demonstrates that the representative score is consistent with the users' satisfaction.

4.1.3 Spatial Object Selection (sos) Problem

We are now ready to define the SOS problem.

Definition 1: Spatial Object Selection (sos) Problem Given a set of geospatial objects $O = \{o_1, \dots, o_n\}$ in a region of interest, a distance threshold θ , and an integer k , the SOS problem aims to select a subset of k objects $S \subseteq O$ such that

1. $\text{dist}(o_i, o_j) \geq \theta$ for any $o_i, o_j \in S$, and
2. $\text{Sim}(O, S)$ is maximized.

The first condition guarantees that the users can easily distinguish two close geospatial objects on the map. The second condition ensures that the selected geospatial objects can well represent all geospatial objects in the region. Note that if users want to specify filtering condition on the set of objects O , e.g., objects should contain keyword “president election,” existing database querying engines can be employed to perform the filtering.

Theorem 7: The SOS Problem is NP-hard.

Proof:

We prove the theorem by a reduction from the decision version of the Minimum Dominating Set problem, which is known to be NP-hard. The Minimum Dominating Set (MDS) problem aims to find the minimum number of nodes such that each node in the graph is either selected or a neighbor of the selected nodes. The decision problem is to decide whether there is a solution of no more than k nodes.

Consider a set of geospatial objects, each of which has the same weight. We assume that for each pair of objects o_u and o_v we have $\text{Sim}(o_u, o_v) \in \{0, 1\}$. The distance threshold θ is set small enough such that any pair of objects fulfills the Visibility Constraint. For any instance of the Minimum Dominating Set decision (MDSd) problem, we can build an SOS problem to solve it as follows. Given a graph $G(V, E)$, we map each node u_i to an object o_i . If there is an edge between nodes i and j , we set $\text{sim}(o_i, o_j)$ to be 1, otherwise $\text{sim}(o_i, o_j) = 0$.

1. Given an MDSd problem, if nodes T are the result, we let $S = \{o_i | u_i \in T\}$, and obviously S is the result of the SOS problem.
2. Given the result S of the constructed SOS problem, if we have $|O| = \sum_{o \in O} \text{Sim}(o, S)$, for each object o we know that $\text{Sim}(o, S) = 1$. According to Equation 4.1, we have either $o \in S$ (where $\text{Sim}(o, o) = 1$) or o is represented by S (where

$\text{Sim}_{o' \in S}(o, o') = 1$). Let $T = \{u_i | o_i \in S\}$ be a set of nodes, and T is the result of the MDSd problem, since for each node u we have either $u \in T$ or $(u, v) \in E \& v \in T$ (u is dominated by T).

Therefore, the SOS problem is NP-hard. $\square$

4.1.4 Interactive Exploration on Map

When users navigate the map to explore the geospatial data, they can perform three different navigation operations: (1) zooming in, (2) zooming out, and (3) panning.

Zooming in (out). By zooming in (out), the map is displayed with a finer (coarser) granularity and more (fewer) details in the region are visible to users (while the center of the map remains unchanged).

Panning. Users can move the displayed region to a new place with the same granularity.

With regard to the three operations, in order to provide a seamless experience for the end-user, the map needs to fulfill the *zoom consistency* and the *movement consistency* constraints.

Zooming Consistency Constraint: for any object o appearing at any coarse granularity, it should also appear in all finer granularities of regions containing the location of that object as users zoom the map.

Panning Consistency Constraint: at a certain granularity, for any object o appearing at a region, it should also appear in all other regions which contain the location of this object as user pans the map.

Next we present how the two consistency constraints affect the SOS problem as users navigate the map. We consider the three examples shown in Figure 1.3(a), 1.3(b), and 1.3(c), respectively. Each point in the map is a POI. The set of POIs that are visible to users are marked with red color. Assume that three objects should be selected to display to users in the region of interest.

Example 6: [Zoom in.] Consider the example in Figure 1.3(a). Before the zoom-in operation, region r_1 is displayed to the user. There are nine objects in the map region $(o_1, \dots, o_9)$, in which objects o_1 , o_5 and o_9 are visible to users. After the zoom-in operation, region r_2 is displayed to the user. We need to select three objects from r_2 to display. However, since object o_5 is visible to the user before zooming in, it should still be visible. Thus, we need to select two objects from all objects in r_2 excluding o_5 to display. $\square$

Example 7: [Zoom out.] Consider the example in Figure 1.3(b). Before the zoom-out operation, region r_1 is displayed to the user. There are four objects in the map region $(o_3, \dots, o_6)$, in which o_4 , o_5 and o_6 are visible. After the zoom-out operation, region r_2 is displayed to the user with a coarser granularity. We need to select three objects from r_2 to display. Note that o_6 cannot be selected, since it is not visible in the finer granularity. This means only the black nodes in r_1 can be selected in the new map region. Thus, in the new map region, we need to select three objects from all objects in $r_2 \setminus r_1$ and the black nodes in r_1 . $\square$

Example 8: [Panning.] Consider the example in Figure 1.3(c). Before the panning operation, region r_1 is displayed to the user. There are 7 objects in the map region, in which objects o_5 and o_9 are visible. After the panning operation, region r_2 is displayed to the user. We need to select three objects from r_2 to display. Note that object o_7 cannot be selected, since it is not visible before panning. Moreover, object o_5 should be selected because it is visible before panning. Thus, we need to select two objects from the region $r_2 \setminus r_1$ to display. $\square$

From the three examples, we observe that in the new map region (1) some objects must be included into the representative set, and (2) some objects must be excluded from the representative set. Specifically, in the new map region, there is a set D of geospatial objects that must be selected because of the consistency constraints. In these three examples, the set D are $\{o_5\}$, $\{\}$ and $\{o_5\}$, respectively. There is also a set G of candidate objects only from which we can select objects into the representative set. In the above examples, the set G are $\{o_3, o_4, o_6\}$, $\{o_1, o_2, o_3, o_4, o_5, o_7, o_8, o_9\}$ and $\{o_1, o_3, o_{10}, o_{11}, o_{12}\}$, respectively.

4.1.5 Interactive Spatial Object Selection (isos) Problem

We proceed to introduce the *interactive spatial object selection* (ISOS) problem, which enforces the zooming and panning consistency constraints on top of the SOS problem to support interactive exploration of geospatial data.

Definition 2: Interactive Spatial Object Selection (isos) Problem Given a set of geospatial objects $O = \{o_1, \dots, o_n\}$ in a region, a distance threshold θ , and an integer k , let $G \subseteq O$ be the set of candidate geospatial objects, $D \subseteq O$ be the set of geospatial objects that should remain visible to users after users perform any of the three navigation operations according to the zooming consistency and panning consistency. The ISOS problem aims to select a subset $S \subseteq G$, $|S \cup D| = k$, such that

1. $\text{dist}(o_i, o_j) \geq \theta$ for any $o_i, o_j \in S \cup D$, and
2. $\text{Sim}(O, S \cup D)$ is maximized.

4.2 Proposed Algorithm for SOS

In this section we present the proposed greedy algorithm for finding an approximate solution to the SOS problem, and we prove that the proposed algorithm has an approximation ratio of $1/8$.

4.2.1 A Greedy Algorithm

The intractability result motivates us to develop a greedy algorithm for the SOS problem. To select a set S of geospatial objects from the set O of objects in a greedy manner, we need to take both the representative constraint and the visibility constraint into consideration. To choose representative objects, the geospatial object that we greedily select should have a high marginal similarity increase. To impose the visibility constraint, the distance between the new selection and any previously selected object should be no less than the given threshold. Specifically, in the proposed greedy algorithm, we select the representative set of objects iteratively. In each iteration, we select the geospatial object with the maximum marginal similarity increase. Then we remove the remaining geospatial objects that do not satisfy the visibility constraint, i.e., its distance to the newly-selected geospatial object is less than the given threshold. The algorithm terminates when k geospatial objects are selected.

One main challenge here is how to efficiently find the object with the maximum marginal similarity increase in each iteration. One naive idea is to compute the marginal increase for each geospatial object. However, this is prohibitively expensive. To address this challenge, we utilize a “lazy-forward” strategy based on the following lemma.

Lemma 4:(Submodularity.) Let S and T be two sets of geospatial objects, and $S \subseteq T$. Let v be a newly inserted object. We have $\text{Sim}(O, S \cup \{v\}) - \text{Sim}(O, S) \geq \text{Sim}(O, T \cup \{v\}) - \text{Sim}(O, T)$.

Proof:

We first prove that

$$\text{Sim}(o, S \cup \{v\}) - \text{Sim}(o, S) \geq \text{Sim}(o, T \cup \{v\}) - \text{Sim}(o, T).$$

We consider the following cases:

Case 1: $Sim(v, o) \leq Sim(o, S)$, $Sim(v, o) \leq Sim(o, T)$. Then $Sim(o, S \cup \{v\}) = Sim(o, S)$ and $Sim(o, T \cup \{v\}) = Sim(o, T)$.

Case 2: $Sim(v, o) > Sim(o, S)$, $Sim(v, o) \leq Sim(o, T)$. Then $Sim(o, T \cup \{v\}) - Sim(o, T) = 0$ and $Sim(o, S \cup \{v\}) - Sim(o, S) > 0$.

Case 3: $Sim(v, o) > Sim(o, S)$, $Sim(v, o) > Sim(o, T)$. In this case, $Sim(o, S \cup \{v\}) = Sim(o, v)$ and $Sim(o, T \cup \{v\}) = Sim(o, v)$. Since $Sim(O, S) \leq Sim(O, T)$, we have $Sim(o, v) - Sim(O, S) \geq Sim(o, v) - Sim(O, T)$.

Since $Sim(O, S) = \frac{1}{|O|} \sum_{o \in O} o.\omega \times Sim(o, S)$, the lemma is proved. $\square$

Since we select one object in each iteration, Lemma 4 shows that the marginal similarity increase of an object (corresponding to v in Lemma 4) in current iteration (corresponding to T) is not larger than the marginal similarity increase in the previous iteration (corresponding to S). Therefore, based on this lemma, to avoid massive recomputation we utilize the “lazy forward” strategy, which works as follows: For each geospatial object o , we construct a tuple $t = \langle o, \Delta(o), Iter \rangle$, where $\Delta(o) = Sim(O, S \cup \{o\}) - Sim(O, S)$ is the marginal similarity increase caused by adding object o to the current representative set S , and $Iter$ records the iteration in which $\Delta(o)$ is computed. We use a max-heap to maintain the tuples for all spatial objects according to $\Delta(o)$. In the i -th iteration, we check the top tuple t in the heap. If $\Delta(o)$ of tuple t is not computed in this iteration, i.e., $t.Iter \neq i$, the value $\Delta(o)$ is not up-to-date. Note that $\Delta(o)$ can serve as an upper bound for its real marginal increase according to Lemma 4. Therefore, we need to recompute its marginal increase and push it back to the heap. We keep checking the top tuple until the marginal increase of the top tuple is computed in this iteration ($t.Iter = i$)—The real marginal increase of the geospatial object of the top tuple is higher than the upper bound marginal increase of the other geospatial objects. Therefore, the corresponding geospatial object is picked into the representative set.

Note that in the “lazy forward” strategy, we recompute the marginal increase only for those objects appearing as the top tuple in the heap, rather than for all geospatial objects. For those objects whose marginal increases are computed in previous iterations, their values become outdated, but they can still serve as upper bound values for the marginal increase in the current round. This guarantees the correctness of the “lazy forward” strategy.

The details of the algorithm are shown in Algorithm 5. It takes as input a set O of geospatial objects, the size of the representative set k , and a distance threshold θ .

Its output is the set of selected objects S , and it is initialized as an empty set (line 1). First, the algorithm initializes the max-heap with tuples (lines 2–3). Specifically, for each geospatial object o , it pushes a tuple $\langle o, \text{Sim}(O, \{o\}), 0 \rangle$ into the heap, where $\text{Sim}(O, \{o\})$ is the marginal similarity increase of adding o to S (S is empty set now). Then, the algorithm selects the representative set iteratively (lines 4–12). In each iteration, the algorithm keeps checking the top tuple from the heap (line 6). If the marginal increase of the top tuple t is not computed in this iteration, we need to recompute its marginal increase (line 7) and update $t.\text{Iter}$ (line 8) by the current iteration number. We then push t back to the heap. When the marginal increase of the top tuple is computed in current iteration, we select the corresponding object $t.o$ into the representative set (line 10). In addition, for any geospatial object o' whose distance to o is smaller than the threshold, we remove the corresponding tuple from the heap due to the visibility constraint (lines 11–12).

Example 9: Consider the example shown in Figure 4.1. The collection of geospatial data comprise six geospatial objects o_1, o_2, o_3, o_4, o_5 and o_6 . The similarity between any two objects is given in the table on the right side. We assume that every object has a weight of 1. We want to select a set of two geospatial objects to be visualized.

The algorithm first initializes the heap by computing the marginal increase of each object. For example, the marginal similarity increase of adding object o_1 to $S = \emptyset$ is

$$\Delta(o_1, S) = (1 + 0.9 + 0.2 + 0.5 + 0 + 0) - 0 = 2.6$$

The status of the heap after the initialization is shown in Figure 4.2(a).

In the first iteration, the top tuple in the heap is $t = \langle o_1, 2.6, 0 \rangle$. since $t.\text{Iter} = 0$, the marginal increase of o_1 is computed in this iteration. Therefore we select o_1 into the representative set S . Moreover, we need to remove all objects that conflict with o_1 from the heap due to the visibility constraint. We use a dashed circle to denote the region in which the objects conflict with o_1 . Since $\text{dist}(o_1, o_2) < \theta$, $\text{dist}(o_1, o_5) < \theta$, o_2 and o_5 are removed from the heap. There are only three tuples left in the heap. The status of the heap is shown in Figure 4.2(b).

In the second iteration, the top tuple in heap is $t = \langle o_4, 2.5, 0 \rangle$, and its marginal increase is not updated in this iteration. Therefore we need to recompute the marginal increase as follows:

$$\begin{aligned} & \Delta(o_3, S) \\ &= (1 + 0.9 + 1 + 0.9 + 0 + 0) - (1 + 0.9 + 0.2 + 0.5 + 0 + 0) \\ &= 1.2 \end{aligned} \tag{4.3}$$

Figure 4.1: Example for the greedy algorithm.

Figure 4.2: Status of the max-heap.

Similarly, we next recompute the marginal increase of o_4 as

$$\begin{aligned}
 \Delta(o_4, S) &= (1 + 0.9 + 1 + 0.9 + 0 + 0) - (1 + 0.9 + 0.2 + 0.5 + 0 + 0) \\
 &= 1.2
 \end{aligned} \tag{4.4}$$

In this iteration, we have recomputed the marginal increase for two objects, and the status of the heap is shown in Figure 4.2(c). Note that since the current upper bound for object o_6 is smaller than that of the other two objects, its marginal increase will not be recomputed. With the “lazy forward” strategy, we prune one recomputation in this iteration. Then o_4 is selected into the heap since it has the maximum marginal increase. Now we have successfully selected two objects into the representative set. $\square$

Algorithm 5: Greedy

---

input : A set of geospatial objects O , size k , distance threshold θ
output: A subset of objects S

```

1  $S \leftarrow \emptyset$ ;
2 foreach object  $o$  in  $O$  do
3   | Push  $\langle o, \text{Sim}(O, \{o\}), 0 \rangle$  into heap  $H$ ;
4 while  $|S| < k$  and  $H$  is not empty do
5   |  $t \leftarrow$  pop the top tuple from heap  $H$ ;
6   | while  $t.\text{Iter} \neq |S|$  do
7     |  $t.\Delta(o) \leftarrow \text{Sim}(O, S \cup \{o\}) - \text{Sim}(O, S)$ ;
8     |  $t.\text{Iter} = |S|$ ;
9     | Push  $t$  to heap  $H$ ;
10  |  $t \leftarrow$  pop the top tuple from heap  $H$ ;
    | // select  $o$  as result
11  |  $S \leftarrow S \cup \{t.o\}$ ;
12  | foreach object  $o'$  s.t.  $\text{dist}(o', o) \leq \theta$  do
13    | remove  $o'$  from  $H$ ;
14 return  $S$ ;
    
```

Complexity It takes $O(n)$ time to recompute the marginal similarity increase for a geospatial object, where n is the number of geospatial objects in O . Let n_c be the number of geospatial objects whose marginal increases are recomputed in the k iterations. The time complexity of the greedy algorithm is $O(n_c \cdot n)$. In practice, n_c is much smaller than n .

4.2.2 Approximation Ratio Analysis

We proceed to analyze the approximation ratio of the proposed greedy algorithm. We first present two lemmas that will be leveraged to prove the approximation ratio.

Lemma 5: Let S and T be two sets of geospatial objects, and $S \subseteq T$. We have

$$\text{Sim}(O, S) \leq \text{Sim}(O, T) \quad (4.5)$$

Proof:

Since $S \subseteq T$, we have $\max_{o' \in S} \text{Sim}(o, o') \leq \max_{o' \in T} \text{Sim}(o, o')$. Therefore, we have $\text{Sim}(O, S) \leq \text{Sim}(O, T)$. □ □

Lemma 6: Let S be a set of geospatial objects that satisfy the visibility constraint. Let o be a geospatial object and $o \notin S$. At most 7 objects in S conflict with o .

Proof:

If an object $v \in S$ conflicts with o , then v must be inside the circle with a radius θ centered at o . Since S satisfies the visibility constraint, any two objects in S do not conflict with each other, i.e., $\text{dist}(v_1, v_2) > \theta$ for $v_1, v_2 \in S$. Thus, the circle with a radius of θ centered at o can cover at most 7 objects from S , which is illustrated in Figure 4.3. Note that o is at the position of o_1 . $\square$

Figure 4.3: Conflict Objects Example

With the aforementioned lemmas, we are ready to prove the approximation ratio of the greedy algorithm.

Theorem 8: The Greedy algorithm has an approximation ratio of $1/8$.

Proof:

Let $S^* = [v_1^*, \dots, v_k^*]$ be the optimal set of k objects with maximum score. Let $S = [v_1, \dots, v_k]$ be the set of k selected by **Greedy**. We use $\Delta(v|S) = \text{Sim}(O, S \cup \{v\}) - \text{Sim}(O, S)$ to denote the increase of the score brought by adding v into S . According to Lemma 5 and Lemma 4, we have

$$\begin{aligned}
 \text{Sim}(O, S^*) &\leq \text{Sim}(O, S^* \cup S) \\
 &= \text{Sim}(O, S) + \sum_{j=1}^k \Delta(v_j^* | S \cup \{v_1^*, \dots, v_{j-1}^*\}) \\
 &\leq \text{Sim}(O, S) + \sum_{v^* \in S^*} \Delta(v^* | S)
 \end{aligned} \tag{4.6}$$

Let v^* be an object in S^* and θ be the distance threshold. For an object o , we refer to the circle with a radius of θ centered at o as the **circle**. We consider the following cases:

Case 1: Object v^* does not fall into the circle of any object in S , i.e., $\text{dist}(v^*, v) > \theta$ for any $v \in S$.

When selecting the k -th object in S , v^* also satisfies the Visibility Constraint. Since S is greedily selected, we have

$$\text{Sim}(O, S) - \text{Sim}(O, S_{k-1}) \geq \Delta(v^*|S_{k-1}),$$

where $S_{k-1} = [v_1, \dots, v_{k-1}]$.

According to Lemma 5, we have $\Delta(v^*|S_{k-1}) \geq \Delta(v^*|S)$. Then we can derive $\text{Sim}(O, S) - \text{Sim}(O, S_{k-1}) \geq \Delta(v^*|S)$.

Case 2: Object v^* does not fall into the circles of objects $v_1, \dots, v_i$, but falls into the circle of v_{i+1} , i.e., $\text{dist}(v^*, v_j) > \theta$ for $j \in [1, i]$, and $\text{dist}(v^*, v_{i+1}) \leq \theta$.

When selecting the object v_{i+1} , v^* satisfies the visibility constraint. Since S_{i+1} is greedily selected, we have

$$\text{Sim}(O, S_{i+1}) - \text{Sim}(O, S_i) \geq \Delta(v^*|S_i)$$

According to Lemma 5, we have $\Delta(v^*|S_{i+1}) \geq \Delta(v^*|S)$. Then we can derive $\text{Sim}(O, S_{i+1}) - \text{Sim}(O, S_i) \geq \Delta(v^*|S)$.

From the two cases, we can see that for each object $v^* \in S^*$, we can find an $v_i \in S$ such that v^* and v_i are conflicted and $\text{Sim}(O, S_{i+1}) - \text{Sim}(O, S_i) \geq \Delta(v^*|S)$. Since S^* satisfies the visibility constraint, according to Lemma 6, we have

$$\begin{aligned} \Delta(v^*|S_i) &\leq 7(\text{Sim}(O, S_1) - \text{Sim}(O, S_0) + \dots \\ &\quad + \text{Sim}(O, S) - \text{Sim}(O, S_{k-1})) \\ &= 7\text{Sim}(O, S) \end{aligned}$$

From Equation 4.6, we have

$$\text{Sim}(O, S^*) \leq \text{Sim}(O, S) + 7\text{Sim}(O, S) = 8\text{Sim}(O, S) \quad (4.7)$$

Thus, the Greedy has an approximation ratio of $1/8$. $\square$

4.3 Proposed Algorithm For ISOS

In this section, we first present the greedy algorithm for the ISOS problem, and then propose a pre-fetching strategy to accelerate the processing for the ISOS problem.

4.3.1 A Greedy Algorithm

The ISOS problem extends the SOS problem in two aspects: (1) The ISOS problem selects objects from the candidate set G , while the SOS problem selects from all objects in the region of interest. (2) The selected objects and the pre-determined set of objects D together form the representative set in the ISOS problem, while the representative set in the SOS problem consists of the selected objects only. We can slightly extend the greedy algorithm to solve the ISOS problem, and note that the ISOS problem has the same approximation ratio as that for the SOS problem.

The main idea of the greedy algorithm is still the same. In each iteration, we select the geospatial object with the maximum marginal similarity increase. To make the greedy algorithm work for the ISOS problem, we make the following changes: (1) Since the representative set S must contain all objects in the pre-determined set D , we initialize the set S with all geospatial objects in D , i.e., $S \leftarrow D$ in line 1 in Algorithm 5. (2) Because we can only select geospatial objects from the candidate set G , we initialize the heap H by only using the objects in G . In line 2 of Algorithm 5, we compute the marginal increase for each object $o \in G$.

With the two modifications, the greedy algorithm can be used to find a set of representative geospatial objects for the ISOS problem.

4.3.2 Pre-fetching Strategy

We proceed to present the new idea of using pre-fetching to speed-up the greedy algorithm. When an end-user explores the geospatial data on the map, she may (1) check the content in a displayed map region, (2) perform a navigation operation, like zoom-in, zoom-out, and panning, and then (3) wait for the visualized exploration system rendering the set of representative geospatial objects on the map. User may repeat such kind of exploration several times until she is satisfied with the results. To reduce the waiting time of the user and provide the user with a seamless browsing experience, we propose to pre-fetch and pre-compute some useful information while the user is still in step 1. In our work, we focus on how to utilize the pre-fetched data to accelerate the greedy algorithm for the ISOS problem. We are not solving the problem of predicting the user's next region of interest, which is addressed by Leilani et al. [BCS16]. In fact, this work is complementary to our work, and can be employed to predict what region of data to pre-fetch.

The main challenges in designing a pre-fetching strategy include (1) what kind of information should we pre-fetch and pre-compute? and (2) how the information can be used to accelerate the selection of representative set?

To address the aforementioned challenges, we first analyze the bottleneck of the greedy algorithm. Because the “lazy forward” strategy can greatly speed up the subsequent computation of marginal similarity increase, the bottleneck of Algorithm 5 is the initialization of the heap. In the initialization, we need to compute the marginal increase of representative score for each object in G , and push the tuple for each object into the heap. This initialization takes $O(n \cdot |G|)$ time, where n is the number of objects in the map region, and $|G|$ is the number of objects in the candidate set. A natural idea is that if we can estimate an upper bound for the marginal increase for each object, we can use the “lazy forward” strategy in the first iteration in the greedy selection.

Next, we present a pre-fetching strategy for estimating the upper bound of the marginal increase for each object in the initialization of the heap, in the context of the three navigation operations, respectively.

4.3.2.1 Pre-fetching for Zoom-in

Recall that after the zoom-in operation, the new region of interest on the map is inside the old region of interest. Figure 4.4 shows a zoom-in example, where r_p is the old region of interest, and r_n is a possible new region of interest after the zoom-in operation.

Let O_p be the set of objects in r_p , and O_n be the set of objects in the new region r_n of interest. Let D be the set of pre-determined geospatial objects in the new region r_n of interest according to the zooming consistency constraint. To accelerate the greedy algorithm, we estimate the marginal increase of representative score for each object o in $O_n \setminus D$, i.e., $Sim(O_n, D \cup \{o\}) - Sim(O_n, D)$ by the following lemma.

Lemma 7: We have $Sim(O_n, D \cup \{o\}) - Sim(O_n, D) \leq \sum_{o' \in O_p} Sim(o, o')$.

Proof:

According to Lemma 4, we have $Sim(O_n, D \cup \{o\}) - Sim(O_n, D) \leq Sim(O_n, \{o\})$. Since $O_n \subseteq O_p$, we have $\sum_{o' \in O_n} Sim(o', o) \leq \sum_{o' \in O_p} Sim(o', o)$. Therefore, $\sum_{o' \in O_p} Sim(o, o')$ is the upper bound of the marginal increase for object o . $\square$

According to Lemma 7, for any new region of interest r_n as a result of zoom-in from r_p , we pre-compute the upper bound of marginal increase of representative score for every object in O_p . Then, once the zoom-in operation is performed, we can find the geospatial objects in the new region of interest and obtain their upper bounds in $O(1)$ time.

Figure 4.4: Zoom-in Example

Figure 4.5: Zoom-out Example

4.3.2.2 Pre-fetching for Zoom-out

Recall that after the zoom-out operation, the new region of interest on the map must contain the old region of interest. Figure 4.5 shows a zoom-out example, where the region r_p is the old region of interest and region r_n is one possible map region after the zoom-out operation. Since r_n must contain region r_p , let r_A be the union of all possible new regions of interest.

Let O_n be the set of objects in the new region of interest, and O_A be the set of objects in the union of all possible new regions of interest. Let G be the candidate set of geospatial objects in the new region of interest according to the zooming consistency constraint. To accelerate the greedy algorithm, we estimate the marginal increase of representative score for each object $o \in G$, i.e., $Sim(O_n, \{o\})$ in the new map region as follows.

Lemma 8: We have $Sim(O_n, \{o\}) < \sum_{o' \in O_A} Sim(o, o')$.

Proof:

According to Lemma 4, we have $Sim(O_n, \{o\}) < Sim(O_A, \{o\})$. According to the definition of the representative score, we have $Sim(O_A, \{o\}) = \sum_{o' \in O_A} Sim(o, o')$. $\square \square$

According to Lemma 8, we can pre-compute the upper bound of the marginal increase for each object o in the union of all possible new regions of interest. Then, once the zoom-out operation is performed, we can find the objects in the new map region and obtain their upper bounds of the marginal increase in $O(1)$ time.

Figure 4.6: Panning Example

4.3.2.3 Pre-fetching for Panning

Recall that after the panning operation, the new region of interest has the same size as the old region of interest on the map. Figure 4.6 shows a panning example, where r_p is the old region of interest before panning, and r_n is one possible new region of interest after panning. Since r_n must overlap with r_p , the union of all possible new regions of interest is r_A , as shown in Figure 4.6.

Let O_n be the set of objects in a new region of interest, and O_A be the set of objects in the union of all possible new regions of interest. Let G and D be the set of candidate objects and the set of pre-determined objects according to the panning consistency constraint, respectively. We estimate the upper bound of the marginal increase of representative score for each object $o \in G$, i.e., $Sim(O_n, D \cup \{o\}) - Sim(O_n, D)$ as follows.

Lemma 9: For an object o , we draw a square r_o centered at o with a width twice of the old region of interest, as shown in Figure 4.6. Let O_r be the set of geospatial objects in the overlapped region $r_A \cap r_o$. We have $Sim(O_n, D \cup \{o\}) - Sim(O_n, D) \leq \sum_{o' \in O_r} Sim(o, o')$

Proof:

The new region of interest is inside the overlapped region $r_A \cap r_o$, thus $O_n \subseteq O_r$. The proof follows the proof for Lemma 8 □ □

According to Lemma 9, we can pre-compute the upper bound of marginal increase of representative score for every object o in O_A . Once the panning operation is performed, we can find the geospatial objects in the new region of interest and obtain their upper bounds in $O(1)$ time.

4.4 A Sampling Extension

We propose a sampling-based method to improve the efficiency of our algorithm, which is especially useful when the size of geospatial objects $|O|$ is large. We prove that with a very high probability this method returns a solution with a very small error ϵ compared to the optimal one. We introduce the sampling extension for the SOS problem for simplicity, and similar conclusion can be easily extended for the ISOS problem.

4.4.1 A Sampling Method

When the number of candidates in O is large, it is time-consuming to obtain a result for SOS problem even with our proposed Greedy algorithm, because computing the similarities or testing the Visibility Constraint alone will cost $O(n^2)$ time in the worst case. To tackle this problem, our idea is to sample a small set of objects O' , such that the characteristics of O' are similar to those of O . Ideally, if we apply our Greedy Selection algorithm to objects O' , the selection result can represent O as well, while satisfying the Visibility Constraint. We denote this algorithm by SASS and it is shown in Algorithm 6. The challenge is how to determine a proper size of O' . We will address this in the next subsection, and we will show that with a high probability the result of SASS provides theoretical guarantees on the error bound of the representative score.

Algorithm 6: Sampling for Spatial Object Selection(SaSS)

---

input : A set of geospatial objects O , size k , distance threshold θ , confidence δ , error tolerance ϵ
output: A subset of objects S

- 1 $m = \left\lceil \frac{1}{\frac{2\epsilon^2}{\ln \frac{2}{\delta}} + \frac{1}{|O|}} \right\rceil$;
- 2 Draw m samples O' from O randomly;
- 3 $S \leftarrow \text{Greedy}(O', k, \theta)$; // invoking Algorithm 5
- 4 **return** S ;

Note that simply sampling k objects as the result of the SOS problem is not desirable due to the following reasons. The random sampling does not take into account representativeness and visibility constraints. Even if the visibility constraint can be easily enforced in the random sampling method as we do in experiments (Section 4.5), the sampling results are not representative as shown in our results. We show in Section 4.5 that such a random selection strategy results in a poor representative score.

4.4.2 Determining Sampling Size

Before the proof of theorem, we first introduce some useful concentration inequalities that will be utilized for the proof.

Lemma 10: (Hoeffding's Inequality) [Hoe63] Given a set of n random variables $\mathcal{X} = x_1, x_2, \dots, x_n$ in $[0, 1]$ with a mean μ , for any $\epsilon > 0$, we have

$$\mathbb{P} \left[\left| \frac{1}{n} \sum_{i=1}^n x_i - \mu \right| \geq \epsilon \right] \leq 2\exp(-2\epsilon^2 n). \quad (4.8)$$

□

Lemma 10 provides a general bound for infinite population, i.e., the number of variables n . In most specific cases, for a finite population there exists a tighter bound:

Lemma 11: (Serfling's Inequality) [Ser74] Given a set of n random variables $\mathcal{X} = x_1, x_2, \dots, x_n$ in $[0, 1]$ with a mean μ , for any $\epsilon > 0$ and $1 \leq k < n$, we have

$$\mathbb{P} \left[\max_{k \leq m \leq n-1} \left| \frac{1}{m} \sum_{i=1}^m x_i - \mu \right| \geq \epsilon \right] \leq 2\exp\left(-\frac{2k\epsilon^2}{1 - \frac{k-1}{n}}\right). \quad (4.9)$$

□

Note that both Lemmas 10 and 11 can provide a confidence δ . When the error bound ϵ is large, the confidences are close. In other cases, Equation 4.9 provides a smaller size for sampling. For simplicity, we use Equation 4.8 for the proof, and similar results can be derived by replacing it with Equation 4.9.

Note that the **Greedy** method in Algorithm 6 can be replaced by any other methods for solving SOS problem and we have the following Theorem:

Theorem 9: With probability at least $1 - \delta$, for any approach $\mathcal{F}$ used to solve the SOS problem our SASS returns an $(1 - \epsilon)$ -approximate solution w.r.t. the solution computed by $\mathcal{F}$.

Proof:

The SOS problem aims to maximize $\frac{1}{|O|} \sum_{o \in O} o.\omega \times \text{Sim}(o, S)$. Since $o.\omega \in [0, 1]$ and $\text{Sim}(o, S) \in [0, 1]$, we can take $o.\omega \times \text{Sim}(o, S)$ as a random variable in $[0, 1]$ w.r.t. object o . We use OPT to denote the optimal representative score $\text{Sim}(O, S)$ for a given set of objects O . According to the law of large numbers [Fel08], when O is large, we can simply assume that $OPT = \frac{1}{|O|} \sum_{o \in O} o.\omega \times \text{Sim}(o, S) = \mu$. Let $x_i = o_i.\omega \times \text{Sim}(o_i, S)$, and we

take Equation 4.2 into Equation 4.8: $\mathbb{P}[|Sim(O', S) - OPT| \geq \epsilon] \leq \delta = 2exp(-2\epsilon^2|O'|)$. So we have: $\mathbb{P}[|Sim(O', S) - OPT| < \epsilon] \geq 1 - \delta$, and

$$|O'| = \min \left\{ \left\lceil \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} \right\rceil, |O| \right\} \quad (4.10)$$

□

Similarly, we can obtain the size of objects to be sampled using Eq 4.9, and we have:

$$|O'| = \left\lceil \frac{1}{\frac{2\epsilon^2}{\ln \frac{2}{\delta}} + \frac{1}{|O|}} \right\rceil, \quad (4.11)$$

when $|O| \rightarrow \infty$ we can see that it is identical to Eq 4.10.

4.5 Experiments

We conduct extensive experimental evaluation on the efficiency and effectiveness of our solutions to the Spatial Object Selection (SOS) problem and the Interactive Spatial Object Selection (ISOS) problem. We first introduce the experimental setup, and then report the efficiency of our algorithms and the representative scores, which reflect the effectiveness of the proposed solutions.

4.5.1 Experimental Setup

Datasets. We conduct the experiments on two types of real-life geospatial datasets:

1) **Twitter** is a geo-tagged tweet dataset crawled using Twitter API¹. We use tweets posted by users in United Kingdom (UK) and United States (US), denoted by **UK** and **US**, which contain up to 2 million and 200 million geo-tagged tweets, respectively. Unless specified otherwise we extract 1 million tweets of **UK** and 100 million tweets of **US** for all experiments except for scalability evaluation (in Sec. 4.5.3.6).

2) **POI** is a Point-of-Interest dataset crawled using Foursquare API². Each POI is associated with textual description. It contains 322,006 POIs in Singapore (SG).

For each geospatial object, we randomly set the weight ω in $[0, 1]$.

Algorithms. For the SOS problem, as mentioned in Sections 1.3.2 and 2.2, this is the first work that takes both representative constraint and visibility constraint into consideration

---

¹<https://dev.twitter.com/rest/public>

²<https://developer.foursquare.com/>

in k -size object selection and accepts a general definition of the representative score between selected objects and the whole objects (within a particular region). Therefore, there exists no previous work for direct comparison.

We consider a baseline method **Random** to compare with our **Greedy** method (proposed in Sec. 4.2.1). **Random** is a random selection strategy. To meet the visibility constraint, we repeatedly pick a random object o if adding o into the current result does not break the visibility constraint. Once there are k objects, they are returned as result. We also compare three other baselines, which are introduced in Section 4.5.2. **MAXMIN** and **MAXSUM** [DP14] are two methods for selecting the most diverse objects, and **K-means** is used for clustering. Note that the results of these three methods may not fulfill the visibility constraint.

Performance Measurement. For all the methods, we use R-tree as the spatial index for region queries, which returns all the objects inside a given query region. We evaluate the performance of all methods by their runtime, and we report the runtime after the object fetching is finished. Specifically, for SASS, we also study how many objects are sampled, i.e., $|O'|/|O|$. Given a sampling selection result S , we can compute its representative score on the whole dataset O . We report the score differences, i.e., $|Score(O, S) - Score(O', S)|$. This result is to show how well the result on the sampled objects can represent the whole dataset. Each experiment is repeated 50 times, and the average performance is reported.

Similarity Metric. Note that our approach is able to deal with any similarity metric depending on the richness of data in particular application scenarios. For the **Twitter** and **POI** datasets, each object is associated with some keywords. We measure the similarity of two objects with Cosine Similarity of the keyword vectors.

Query Generation. For the SOS problem, we randomly pick an object from the dataset and generated a square-shape query region R centered at this object. For the ISOS problem, each interactive operation will result in a new region that the user may further explore. For the Zoom-in operation, we randomly locate a new square-shape query region R_{in} that is completely inside the previous region R . For the Zoom-out operation, we randomly locate a new square-shape query region R_{out} that completely covers the previous region R .

Parameters. Table 4.1 shows the detailed settings of all parameters, where the default one is highlighted in bold. By default, we set the query region R as 0.01 of the size of the whole dataset, which usually represents a suburb. It is a reasonable setting as most

Table 4.1: Parameters

Parameter	Range
Query Region Size ($\ast 10^{-2}$)	0.6, 0.8, 1 , 1.2, 1.4
Number of Selected Objects k	60, 80, 100 , 120, 140
Distance threshold θ ($\ast 10^{-3}$)	1, 2, 3 , 4, 5
Zoom-in Size R_{in}/R (by length)	0.3, 0.4, 0.5 , 0.6, 0.7
Zoom-out Size R_{out}/R (by length)	1.2, 1.4, 1.6, 1.8, 2.0
Upscale of data size for scalability test	1 , 1.25, 1.5, 1.75, 2
Relative Error Bound ϵ ($\ast 10^{-2}$)	3, 4, 5 , 6, 7
Confidence Error δ	0.08, 0.09, 0.1 , 0.11, 0.12

users explore a suburb at a time. In average, there are about 0.5% objects of the whole data in a query region of size 0.01, and up to about 15% objects in some dense regions.

In each query region, we select $k=100$ objects by default out of all the objects. For the visibility constraint, we set a distance threshold θ as 0.003 of the size of the query region by length. We set the relative error bound ϵ as 0.05 and the confidence error δ as 0.1 for SASS. We set R_{in} as half of that of R by length, and set R_{out} as two times of R by length. For the panning operation, we randomly locate the new query region R_{pan} that intersects with R , and we assume they are of the same size. We study the impact of overlap ratio of R_{pan} and R on the runtime in Section 4.5.4.5.

Setup. All algorithms are implemented in C++ complied with GCC 4.8.2 and run on Linux. All algorithms are implemented in C++ and run on Linux with a 2.66GHz CPU and 64GB RAM.

4.5.2 A User Study on the Representative Score

To demonstrate the usefulness of the representative score function, we compare with four other selection algorithms, which represent the best baselines we can come up with, and ask 15 students to rate the representative quality. We use Euclidean distance as the similarity metric, which is straightforward to be judged by users, and we assume each object has the same weight. Figure 4.7 shows the selection results on the Twitter dataset in UK, where each method selects 30 objects out of 500. Figure 4.7(a) shows the original distribution of all the objects on the map, and the selection result by our proposed method **Greedy** is shown in Figure 4.7(b). Figure 4.7(c) shows a set of randomly selected objects. Figures 4.7(d) and 4.7(e) show the result of k -DIVERSITY problem[DP14] denoted by $MAXMIN$ and $MAXSUM$, and their objective functions are defined as $f_{MIN}(S) = \max_{S \subseteq O} \min_{\substack{o_i, o_j \in S \\ o_i \neq o_j}} (1 - Sim(o_i, o_j))$, and $f_{SUM}(S) = \max_{\substack{S \subseteq O \\ |S|=k}} \sum_{\substack{o_i, o_j \in S \\ o_i \neq o_j}} (1 - Sim(o_i, o_j))$.

Figure 4.7: Selection Result by Different Methods

Figure 4.7(f) shows the result of k-means clustering, where for each cluster we select the object which is the closest to the cluster centroid. Note that methods *MAXMIN*, *MAXSUM* and k-means may not fulfill the visibility constraint, so in this user study we focus on the representative constraint while ignoring the visibility constraint. Each student is asked to compare the 500 POIs and the selected POIs by each method in term of tweets' content coverage, and gives a score between 1 and 5 for the representative quality, i.e., how well the selected objects represent the 500 POIs, where 1 is the worst

Table 4.2: User Study Result for sos

Method	Greedy	Random	MaxMin	MaxSum	K-means
RP Score	0.95	0.89	0.86	0.56	0.87
User Vote	4.9	3.6	1.6	1.0	3.0

Table 4.3: User Study Result for ISOS

Method	Greedy	Random	MaxMin	MaxSum	K-means
Zooming-in	0.93	0.85	0.72	0.46	0.83
User Vote	4.8	3.5	1.4	1.0	3.2
Zooming-out	0.91	0.83	0.76	0.53	0.81
User Vote	4.7	3.2	1.5	1.2	2.5
Panning	0.94	0.87	0.78	0.51	0.80
User Vote	4.8	3.7	1.6	1.1	3.1

and 5 is the best.

The average rating result is shown in Table 4.2. We can see our method outperforms other methods. We also compute the representative score using Eq 4.2 for the result of each method. We find that the users' rating is consistent with the Representative Score computed by Equation 4.2, which indicates a general usefulness of our quantification on the representative constraint.

We further study the users' experience on the interactive exploration, i.e., the users' satisfaction after zooming in, zooming out and panning operations. We still compare the algorithms for selecting 30 objects out of 500. However, to support zooming out and panning navigations we shrink the window size by half (compared to the previous study). The students are asked to evaluate the representative quality after each of the three operations. The rating result is shown in Table 4.3. We observe that for the ISOS problem, the user's rating is also consistent with the Representative Score.

4.5.3 Evaluation of Solutions to sos Problem

4.5.3.1 Comparing Different Methods

In this experiment, we use default settings for all parameters to compare the performance of our Greedy and SASS algorithms with the baseline methods in solving the problem.

Figures 4.8 and 4.9 show the runtime and representative score on two smaller datasets, UK and POI. Note that the left y-axis of Figure 4.9 uses logarithmic scale. We notice that the runtime of our proposed algorithm Greedy is nearly the same with that of Random

Figure 4.8: Comparing Different Methods on UK

Figure 4.9: Comparing Different Methods on POI

and is about 2/3 of that of **K-means**, while **Greedy** achieves a much better representative score than all the other baselines. We also observe that **SASS** is a bit faster than **Greedy**, and much faster than all the other baselines (except **Random**), since it works on a small set of sampled objects, while the score is close to **Greedy** and better than the others.

The results of **K-means**, **MAXMIN** and **MAXSUM** do not fulfill the visibility constraint, and **SASS** is the sampling extension of **Greedy** and is not significantly better than **Greedy** on small datasets. Therefore for the rest experiments on the two small datasets, we only show the results of **Greedy** and **Random** on **UK** and **POI**. We evaluate **SASS** on **US**, the larger dataset, on which other methods including **Greedy** are very slow.

4.5.3.2 Varying Sampling Parameters

This experiment is to study how the sampling parameters δ and ϵ affect the performance. The experiment is conducted on the large dataset **US**, and the results are shown in Figures 4.10 and 4.11. Note that we only show the results for **SASS** and **Random** since

all the other methods are much more slower, among which the fastest algorithm, **Greedy**, is slower than SASS by an order of magnitude. We observe that for both errors the ratio of sampled objects increases when the errors increase, and it is shown from Figure 4.10(b) that at most 2% of objects are enough to approximately solve the SOS problem. We also observe from Figures 4.10(a) and 4.11(a) that the runtime decreases, which is because the number of candidate objects decreases. It is observed that the relative score differences are always less than 0.01 when we vary θ or ϵ , which indicates that using sampled data only slightly loses the representativeness.

Figure 4.10: Varying ϵ on US Figure 4.11: Varying δ on US

4.5.3.3 Varying the Query Region Size

This experiment is to study how the query region size affects the performance of the proposed approach. From the result shown in Figure 4.12 we find the runtime of the

proposed approach increases almost linearly when the query region size increases. This is because for a larger query region size, more objects are considered as the input of our algorithms, leading to an increasing runtime. We do not show the runtime of Greedy on US as it is 3 orders of magnitude slower than SaSS. It is because SaSS is based on sampled objects, which is rather efficient even on large datasets, while Greedy works on all objects.

Figure 4.12: Varying query region size

4.5.3.4 Varying the Number of Selected Objects k

This experiment is to study how the number of selected objects, i.e. k , affects the performance. It can be observed from Figure 4.13 that, when k increases the runtime of all algorithms increases. It is because more rounds of iteration are needed for a large k .

 Figure 4.13: Varying k

4.5.3.5 Varying the Distance Threshold

Figure 4.14: Varying distance threshold θ

Figure 4.14 shows the effect of distance thresholds θ on the performance. We observe that the runtime of both algorithms stays stable regardless of the choices of distance threshold.

4.5.3.6 Scalability Test

Figure 4.15: Varying scalability on Twitter

Last, we study the scalability of our proposed algorithms by varying the data size. We vary UK dataset from 1 million to 2 million and US from 100 million to 200 million. It is observed from Figure 4.15(a) that the runtime of Greedy generally increases w.r.t. the

size of the dataset. Since the tweets are from one country, when the number of objects increases, the density of the objects increases in general. Since we fix the query region size, a large dataset usually induces more objects as input and thus the runtime increases. However we observe from Figure 4.15(b) that the runtime of SASS only changes slightly as we vary the number of objects. It is because SASS is based on a certain number of sampled objects. When parameters δ and ϵ are fixed, varying the size of dataset from $100M$ to $200M$ does not change the number of sampled objects significantly.

4.5.4 Evaluation of solutions to isos Problem

Figure 4.16: Pre-fetching vs. non-fetching on UK

This experiment is to study the performance of our solution to solve the Interactive Spatial Object Selection (ISOS) problem. In particular, we need to consider the response time for three interactive operations: Zoom-in, Zoom-out and Panning. We denote our greedy algorithms for each of the three operations by **Greedy-in**, **Greedy-out** and **Greedy-pan**, and they are compared with the algorithms using pre-fetching (proposed in Sec. 4.3), which are denoted by **Pre-in**, **Pre-out** and **Pre-pan** respectively.

First of all, we record the runtime for **Pre-in**, **Pre-out** and **Pre-pan** using default settings for all parameters, and the result is shown in Figure 4.16. We find that the pre-fetching technique improves the efficiency of **Greedy-in**, **Greedy-out** and **Greedy-pan** by almost 2, 1 and 1 order of magnitude respectively, which is very significant.

In the remaining experiments, we deploy the sampling enhanced greedy algorithm, namely SASS on the large dataset **US** for the three operations, and they are denoted by **SASS-in**, **SASS-out** and **SASS-pan**.

Figure 4.17: Varying query region size

4.5.4.1 Varying the Query Region Size

As we can see from Figure 4.17, each method's performance keeps stable when the region size increases, and our pre-fetching method significantly reduces the runtime by 3, 1 and 2 orders of magnitude for Zoom-in, Zoom-out and Panning operations, respectively.

4.5.4.2 Varying the Number of Selected Objects k

Figure 4.18 shows that the runtime for each operator increases when k increases. Also, pre-fetching techniques help improve the performance up to 2 orders of magnitude.

 Figure 4.18: Varying k

4.5.4.3 Varying the Visibility Constraint

Figure 4.19 shows similar trends with its counterpart in addressing the SOS problem.

 Figure 4.19: Varying distance threshold θ

4.5.4.4 Scalability Test

Figure 4.20: Varying scalability

Figure 4.20 shows the runtime while we vary the size of datasets, and we observe similar trends as those in the SOS problem.

4.5.4.5 Zooming Scale and Panning Overlap Study

Figure 4.21(a) presents the performance result w.r.t. a varying zoom-in scale from 0.3 to 0.7. We find that the runtime of Greedy-in scales linearly while Pre-in with scales sub-linearly; also the pre-fetching method helps improving the performance of greedy method by about 2 orders of magnitude. The runtime is less than 10ms for all cases, providing a seamless user experience in interactive exploration. A similar observation can be made for the zoom-out case (Figure 4.21(b)). Last, we study how the overlap rate between two regions before and after the panning operation affects the runtime of our

Figure 4.21: Varying zooming scale and panning overlap on UK

algorithm. As shown in Figure 4.21(c), we have the following observations: (1) when the overlap rate is small, the pre-fetching method can improve the performance by 2 orders of magnitude; (2) when the rate increases to [80%-100%], the significance of pre-fetching reduces accordingly.

4.6 Summary

We have developed an interactive visualized exploration system for geospatial data, which took representativeness, visibility, zooming consistency, and panning consistency into consideration. We first proposed the SOS problem to select top- k representative objects from the current region of interest, and any two selected objects should not be too close to each other for users to distinguish in the limited space of a screen or windows. We proved that it is an NP-hard problem, and developed an approximation algorithm with performance guarantees. To provide a seamless experience for users to interactively explore the data when navigating the map, we formally defined the ISOS problem, and proposed a pre-fetching solution on top of our greedy algorithm to significantly improve the efficiency by almost 2 orders of magnitude. We enhanced the efficiency of the proposed solutions for large datasets by a sampling technique with theoretical guarantee. Our experimental study demonstrated the efficiency and scalability of our approach.

Chapter 5

Spatial Functional Region Exploration

This chapter is organized as follows: Section 5.1 formally defines the Functional Region Segmentation problem. Then in Section 5.2 we introduce three algorithms to solve the problem: a greedy algorithm, an improved greedy algorithm with improved time complexity, and a post-processing algorithm to improve the result. Section 5.3 presents the experiments of all our proposed algorithms. Finally we summarize our contributions of this chapter in Section 5.4.

5.1 Problem Statement

5.1.1 Road Segmentation and Spatial Objects

Given a set of roads R , each road $r \in R$ is described by a sequence of coordinates, $(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_n, lon_n)$. The whole map is partitioned by R into t disjoint regions. We call such region a *block*. Two blocks are called *connected* if they share a common road as the boundary.

Definition 3: Spatial Block.

Given a set of roads R , we can derive a set of blocks B . A block $b \in B$ is enclosed by a set of road segments, and it can be viewed as the set of accessible points, where $b = \{u \mid \text{For any } u, v \in b, \text{ there exists a path from } u \text{ to } v \text{ without passing any road in } R\}$.

It can be derived that for any two points u, v in different blocks, there exists no path from u to v without passing any road in R .

Given a set of spatial objects O , each object $o \in O$ is described by a tuple $\langle lat, lon, cat, value \rangle$, where $\langle lat, lon \rangle$ is the location of the object, cat is the cat-

egory of the object, and *value* is a normalized value in $[0, 1]$ indicating the weight or importance of the object.

Category distribution and aggregation. Given a set of spatial objects O located in a block, we can obtain the category distribution P as follows:

$$P(i)_{i \in Allcat} = \frac{\sum_{o \in O \& o.cat=i} o.value}{\sum_{o \in O} o.value}, \quad (5.1)$$

where *Allcat* is the collection of all object categories.

Given two category distributions P and Q , they can be aggregated as a new distribution $R = Aggr(P, Q)$ as follows:

$$R(i)_{i \in Allcat} = \frac{P(i) + Q(i)}{|P| + |Q|}. \quad (5.2)$$

5.1.2 Information Loss

In this work, we use Kullback-Leibler (KL) divergence [KL51] as the measure of aggregation quality. Formally, KL divergence measures the amount of information that one loses by using a distribution Q to model the components, instead of using the original source distribution P . In our case, we aim to use the aggregated distribution to represent the category distribution in each block.

For a single block, the original category distribution is P . After aggregation with other blocks (will be introduced later), the new category distribution is Q , and the loss of the aggregated distribution Q can be measured as follows:

$$Loss(Q) = D_{KL}(P, Q) = \sum_{i \in Allcat} P(i) \log \frac{P(i)}{Q(i)}. \quad (5.3)$$

Given a set of blocks B and the associated category distributions $P_1, P_2, \dots, P_{|B|}$, we can compute the aggregated category distribution $Q = Aggr(P_1, P_2, \dots, P_{|B|})$ by applying equation 5.2. Then the loss of the aggregated distribution Q can be measured as the loss in each block:

$$Loss(Q) = \frac{1}{|B|} \sum_{j=1}^{|B|} D_{KL}(P_j, Q) = \frac{1}{|B|} \sum_{j=1}^{|B|} \sum_{i \in Allcat} P_j(i) \log \frac{P_j(i)}{Q(i)}. \quad (5.4)$$

Intuitively, the definition of information loss reflects the two features of category distribution aggregation: first, with more blocks are aggregated, the information loss

increases since we aim to use high-level aggregated results to represent the information in low-level blocks; second, aggregating blocks with similar distribution results in low information loss, it is because the aggregated category distribution deviates not too much from the original distributions. In contrast, high information loss will be induced in the result if we merge blocks with dissimilar distributions.

5.1.3 The FRS Problem

Now we are ready to define the Functional Region Segmentation (FRS) problem discussed in this chapter.

Definition 4: Functional Region Segmentation problem.

Given a set of t disjoint blocks on a plane partitioned by roads networks R , and a set of spatial objects O , we aim to segment the whole space by grouping the t blocks into k partitions, where $k(k \leq t)$ is a user-specified parameter. Each partition is called a *Functional Region* if all blocks in a Functional Region are connected. A partitioning result S is called a valid partitioning, if $|S| = k$ and S_i is a Functional Region for all $i \in [1, k]$.

The FRS problem aims to find the partitioning strategy while the following cost among all possible solutions is minimized:

$$InfLoss(S) = \frac{1}{t} \sum_{i=1}^k Loss(S_i) = \frac{1}{t} \sum_{i=1}^k \sum_{j=1}^{|S_i|} D_{KL}(P_{i,j}, S_i), \quad (5.5)$$

where $P_{i,j}$ denotes the category distribution of the j -th block in a Functional Region S_i .

5.1.4 Convert FRS to a Graph Partitioning Problem

We show that the FRS problem can be converted to a graph partitioning problem as follows. Consider an instance of FRS problem shown in Figure 5.1, for each block B_i we assign a node V_i to it. For each pair of connected blocks B_i and B_j , we assign an edge e between nodes V_i and V_j , and we obtain a graph $G(V, E)$. Then the FRS problem is identical to the following graph partitioning problem on G .

Definition 5: Minimum Cost Graph Partitioning (MCGP) problem.

Given a graph $G(V, E)$, the goal of the problem is to partition the graph into k parts, where k is a user-specified parameter. For each partitioning strategy S , we can compute the cost of a partitioning by:

$$Cost(S)_{|S|=k} = \sum_{i=1}^k f(S_i), \quad (5.6)$$

where $f()$ is a monotonic function indicating the cost of merging all nodes inside.

The **MCGP** problem aims to find a valid k -partitioning strategy S on a graph, while the cost function $Cost(S)$ is minimized.

Figure 5.1: Example of Graph Conversion

Theorem 10: The problem of finding a valid partitioning for FRS problem with minimum information loss is NP-hard.

Proof: We prove by reducing the FRS problem to the **MCGP** problem. In particular, we show that a special case of FRS problem is already NP-hard. For the cost function on merging two partitions, we simply assume that the costs are equal on each edge. In this case, the problem of the derived **MCGP** problem is to find a valid k -partitioning strategy S on a graph, while the total cost on the linking edges between partitions is maximized (or minimized if we set negative weights on the edges). It is shown by Bui, et

al [BJ92] that, it is NP-hard to find optimal α -edge separators for general graphs even when restricted to maximum degree 3 graphs.

For any instance of the MCGP problem, we can design a FRS problem to solve it. And thus, we can conclude that the FRS problem is NP-hard. $\square$

5.2 Approaches for FRS Problem

5.2.1 A Bottom-up Greedy Algorithm

In this section, we introduce a greedy solution for the FRS problem. As shown in Section 5.1.4, the FRS problem can be solved by the corresponding converted graph, thus we explain our algorithm using the graph. The problem aims to merge connected and similar nodes into partitions. An intuitive idea to solve the problem is to shrink the size of the graph by merging a pair of connected nodes u and v , which can be viewed as the aggregation of two connected blocks in the FRS problem. After the shrinking, we use a new node to replace the nodes u and v in the graph, while the edges of u and v are updated w.r.t. the new node. This procedure is repeated until we obtain only k nodes in the graph. To minimize the cost function, a greedy strategy for choosing the pair of u and v is to select the pair that can minimize the increase of *information loss* computed by Equation 5.4.

To simplify the procedure of shrinking, however, we do not need to actually repeat the *add* or *delete* operations on the graph. Alternatively, we use a bottom-up merging strategy which is illustrated in Figure 5.2. At first, each node is assigned to a unique partition that contains only one node as shown in Figure 5.2(a). Then, in each step of shrinking, we choose the pair of partitions that increases the least information loss, and the two partitions of nodes are merged into one partition as shown in Figure 5.2(b). The category distributions of two partitions are also aggregated as a new distribution. Note that in each step of shrinking, we only consider the graph from the level of partitions, since the category distribution is associated with each partition. As a result, we only need to merge two partitions instead of merging two set of nodes. Finally, Figure 5.2(c) shows a result of the problem consisting of two partitions.

We notice that in this algorithm we need to frequently combine two partitions, which may take $O(n)$ time in the worst case. To improve the efficiency, we utilize the *union-find*

algorithm proposed by Bernard A. Galler and Michael J. Fischer [GF64], which provides three types of operations: *MakeSet*, *Union* and *Find*. The *MakeSet* operation assigns a set to a node as initialization. The *Union* operation merges two sets, and the *Find* operation returns the set that a node belongs to. It is guaranteed [GF64] that the time complexity of the three operations is up to $O(\log n)$ in the worst case.

Figure 5.2: Example of Graph Shrinking

The pseudo code of the greedy shrinking algorithm is shown in Algorithm 7. We first assign each node to a unique partition using the *MakeSet* operation (Lines 1-2). Then we repeatedly shrink the size of the graph (Lines 3-18), and an *edgeList* is maintained to record the edges left in the graph. When we find a pair of two nodes already in the same partition (Lines 16-17) or two partitions are merged (Lines 18-19), we can safely remove the edge from *edgeList* to reduce redundant computations. In each round of shrinking, we find the pair of nodes *mergeNodeA*, *mergeNodeB* from distinct partitions that minimizes the information loss *minLoss*, and merge the two partitions using *Union* operation (Line 18). Finally, Lines (21-30) organizes the nodes into k partitions and return S as the result.

Time Complexity Analysis. From the pseudo code of Algorithm 7, it is obvious to find that the shrinking terminates in $O(V)$ steps, since in each round of shrinking we merge two partitions into one and the size of the graph decrease linearly until reaching k . Now we analysis the total time complexity. First, in each step of shrinking it costs up to $O(E)$ time to find a pair of partitions with minimum information loss. Second, it costs $O(d)$ time to compute the information loss or the KL divergence of two distributions according to Equation 5.4, where $d = |Allcat|$ is the dimension of categories given in the input. Finally, by combining all together we conclude the overall time complexity of the Algorithm 7 is $O(VE d)$.

Algorithm 7: Greedy()

```
input : A graph  $G$ , the size of partitions  $k$ 
output: A partitioning result  $S$ .
1 foreach node  $u \in V$  do
2   |  $MakeSet(u)$ ;
3  $partNum \leftarrow |V|$ ;
4  $edgeList \leftarrow E$ ;
5 while  $partNum > k$  do
6   |  $minLoss \leftarrow \infty$ ;
7   |  $mergeNodeA, mergeNodeB \leftarrow \emptyset$ ;
8   | foreach edge  $(u, v) \in edgeList$  do
9     |  $setA \leftarrow Find(u)$ ;
10    |  $setB \leftarrow Find(v)$ ;
11    | if  $setA \neq setB$  then
12      |  $cost \leftarrow Loss(setA, setB)$ ;
13      | if  $cost < minLoss$  then
14        |  $minLoss \leftarrow cost$ ;
15        |  $mergeNodeA, mergeNodeB \leftarrow u, v$ ;
16    | else
17      | Remove Edge  $(u, v)$  from  $edgeList$ ;
18    |  $Union(mergeNodeA, mergeNodeB)$ ;
19    | Remove Edge  $(u, v)$  from  $edgeList$ ;
20    |  $partNum \leftarrow partNum - 1$ ;
21 Initialize a map  $Idx$ ;
22  $partId \leftarrow 0$ ;
23  $S \leftarrow \emptyset$ ;
24 foreach node  $u \in V$  do
25   |  $setA \leftarrow Find(u)$ ;
26   | if  $setA$  in  $Idx$  then
27     | Insert  $u$  into  $S[Idx(setA)]$ ;
28   | else
29     |  $Idx(setA) \leftarrow partId$ ;
30     |  $partId \leftarrow partId + 1$ ;
31 return  $S$ ;
```

5.2.2 From Greedy to Greedy⁺

In this section, we propose an enhanced Greedy⁺ algorithm with better time complexity. The Greedy algorithm introduced in Section 5.2.1 terminates in $O(V)$ steps, and our idea to improve the efficiency is to shrink more than one partitions in each round of

iteration. We observe that the FRS problem basically groups the nodes of similar category distribution into the same partition, and if two nodes a and b are both similar to a node c then it is possible that nodes a and b are similar as well. In this way, we can merge two or more nodes into one partition each time.

Figure 5.3: Example of Graph⁺

The idea of the enhanced greedy algorithm is illustrated by Figure 5.3. For each node u , we can compute the information loss by aggregating u and v into one partition for each neighbor v of u . We denote the neighbor v as the *simNeighbor* of u if the information loss w.r.t. node v is the minimum one. In Figure 5.3(a), we find the *simNeighbors* for each node, e.g. $\text{simNeighbor}[A] = B$ and $\text{simNeighbor}[B] = C$. Note that the relation of *simNeighbor* is not necessarily symmetrical, e.g. $\text{simNeighbor}[\text{simNeighbor}[A]] \neq A$. Then we can merge the nodes into one partition if they are connected by the edges of *simNeighbor*. In the example shown in Figure 5.3(a), nodes $\{A, B, C, D\}$ and $\{E, F, G\}$ are grouped, since they are from disjoint components.

The details of the Greedy⁺ are described in Algorithm 8. The main difference of Algorithms 7 and Algorithms 8 is that in Greedy⁺ we find the *simNeighbor* of each node (Lines 9-14) and the partitions are merged directly (Line 15). We show that the time complexity of Greedy⁺ is improved by the following analysis.

Time Complexity Analysis. First, we show that the Greedy⁺ uses fewer rounds of iterations to reach k partitions than Greedy.

Algorithm 8: Greedy⁺()

```

input : A graph  $G$ , the size of partitions  $k$ 
output: A partitioning result  $S$ .
1 foreach node  $u \in V$  do
2   |  $MakeSet(u)$ ;
3  $partNum \leftarrow |V|$ ;
4 while  $partNum > k$  do
5   | foreach node  $u$  in  $V$  do
6     |  $minLoss \leftarrow \infty$ ;
7     |  $simNeighbor \leftarrow \emptyset$ ;
8     |  $setA \leftarrow Find(u)$ ;
9     | foreach neighbor  $v$  of node  $u$  do
10    |   |  $setB \leftarrow Find(v)$ ;
11    |   |  $cost \leftarrow Loss(setA, setB)$ ;
12    |   | if  $setA \neq setB$  and  $cost < minLoss$  then
13    |   |   |  $minLoss \leftarrow cost$ ;
14    |   |   |  $simNeighbor \leftarrow v$ ;
15    |   |  $Union(u, simNeighbor)$ ;
16    |   |  $partNum \leftarrow partNum - 1$ ;
17    |   | if  $partNum \leq k$  then
18    |   |   | Break;
19 Initialize a map  $Idx$ ;
20  $partId \leftarrow 0$ ;
21  $S \leftarrow \emptyset$ ;
22 foreach node  $u \in V$  do
23   |  $setA \leftarrow Find(u)$ ;
24   | if  $setA$  in  $Idx$  then
25   |   | Insert  $u$  into  $S[Idx(setA)]$ ;
26   | else
27   |   |  $Idx(setA) \leftarrow partId$ ;
28   |   |  $partId \leftarrow partId + 1$ ;
29 return  $S$ ;
    
```

Lemma 12: The algorithm Greedy⁺ terminates in $O(\log V)$ steps.

Proof:

Let's consider the shrinking of Greedy⁺ in one round. For each node u we find the $simNeighbor$ v of it, and the two partitions w.r.t. u and v are merged. It is also possible that we find $simNeighbor[i] = u$ for another node i . As a result, we can guarantee that for each partition, it is combined with at least one other partitions. In the worst case, every pair of partitions are merged into one partition, which is also called a *matching* of the

graph and the size of the graph is reduced by half. We can conclude that within $O(\log V)$ steps the graph can be shrunk to k partitions and the Greedy⁺ algorithm terminates. $\square$

Next, we analyze the overall time complexity of Greedy⁺. In each step of the shrinking, it costs at most $O(E)$ time to find the simNeighbor for each node, and it costs $O(d)$ time to compute the information loss or the KL divergence of two distributions, where $d = |Allcat|$ is the dimension of categories given as input. As a result, the overall time complexity of Greedy⁺ is $O(\log VEd)$.

5.2.3 A Refinement Procedure

In this section, we show that the result of the FRS problem can be further improved by a post-processing algorithm. We explain our idea using an example in Figure 5.4.

Figure 5.4: Example of Refinement

In Figure 5.4(a), we first obtain a result of partitioning obtained from either Greedy or Greedy⁺, which already partitions the graph into k parts. We denote the information loss by $infLos_{old}$. Then we try to exclude a node from the partition it belongs, and Figure 5.4(b) shows that node A can be excluded from the left partition. We try to combine this node with other partition, and Figure 5.4(c) shows that node A can be merged with the right partition. Note that it is possible that there are more than one partitions that this excluded node can be assigned to. The information loss in the new result is denoted by $infLos_{new}$. We know that the new result is a better choice if $infLos_{new} < infLos_{old}$, and the procedure can be repeated until we can not find a possible swapping inducing the decrease of information loss.

Algorithm 9: Refine()

```
input  : A partitioning result  $S$ , the size of partitions  $k$ .  
output: A partitioning result  $S$ .  
1  $canSwap \leftarrow True$ ;  
2 while  $canSwap$  do  
3    $canSwap \leftarrow False$ ;  
4    $InfLoss_{old} \leftarrow Loss(S)$ ;  
5   foreach node  $u$  in  $V$  do  
6     foreach neighbor  $v$  of  $u$  do  
7       if  $u, v$  in difference partitions  $S_i, S_j$  then  
8          $S' \leftarrow S$ ;  
9          $S'_i \leftarrow S'_i / u$ ;  
10         $S'_j \leftarrow S'_j \cup u$ ;  
11         $InfLoss_{new} \leftarrow Loss(S')$ ;  
12        if  $InfLoss_{new} < InfLoss_{old}$  then  
13           $S \leftarrow S'$ ;  
14           $canSwap \leftarrow True$ ;  
15          Break;  
16 return  $S$ ;
```

The details of the algorithm are shown in Algorithm 9. In each round, we try to find a pair of nodes u and v belonging to different partitions S_i and S_j . We test if we can transfer node u from partition S_i to S_j while the information loss decreases. The procedure is repeated until no possible transfer can be found. In each round of the refinement, it costs $O(E)$ time to test a possible node transfer and computing the information loss needs $O(d)$. As a result, the overall time complexity of Algorithm 9 is $O(tEd)$, where t denotes the number of transfers used and it depends on the initial partitioning S .

5.3 Experiments

We conduct extensive experimental evaluation on the efficiency and effectiveness of our solutions to the FRS problem. We first introduce the experimental setup, and then report the efficiency and quality of our algorithms.

Table 5.1: Statistics of the dataset

Dataset	#POI	#road networks	#segmented blocks
London	3,483,183	531,870	46,867
Beijing	1,923,386	150,357	16,850
Singapore	334,647	94,378	10,325

5.3.1 Experimental Settings

Datasets. We conduct the experiments on three real-life geospatial datasets from Openstreetmap¹, namely *London*, *Beijing* and *Singapore*. The details of the datasets are shown in Table 5.1. The Openstreetmap provides various types of street map data of a city, e.g. amenities, places and waterways, and we use two types of data for the evaluation: the road networks and POIs. For the POI data, each record contains the type of category, and in total there are 15 types of categories. For each POI, we randomly choose a weight in the range $[0, 1]$.

Algorithms and Performance Measurement. Since there exists no previous work for direct comparison of the FRS problem, we conduct the experiments using our proposed methods, namely the Greedy (introduced in Section 5.2.1), Greedy⁺ (introduced in Section 5.2.2), and we combine the two algorithm with the post-processing optimization introduced in Section 5.2.3, which are denoted by G-Ref and G-Ref⁺, respectively. We evaluate the runtime and information loss for the proposed algorithms.

Parameters. By default, we set the number of target functional regions k to be 9, and we set the size of category distribution d to be 9.

Setup. All algorithms are implemented in C++ complied with GCC 4.8.2 and run on Linux. All algorithms are implemented in C++ and run on Linux with a 2.66GHz CPU and 64GB RAM.

5.3.2 Experimental Results

5.3.2.1 Comparing Different Algorithms

This experiment is to study the performance of the algorithms we proposed. The result is shown in Figures 5.5–5.7. It is observed from all three datasets that the runtime of Greedy⁺ is always better than Greedy, it is because the time complexity of Greedy⁺ algorithm is lower than Greedy. However, the information loss of Greedy is better than

¹<https://mapzen.com/data/metro-extracts/>

Greedy⁺, and a possible reason is that Greedy aims to minimize the cost in each step. We can also observe that after the post-processing, the performance of both Greedy and Greedy⁺ are improved, however the runtime after the refinement is not increased a lot.

Figure 5.5: Comparing Algorithms on London

Figure 5.6: Comparing Algorithms on Beijing

5.3.2.2 Varying the Number of k

This experiment is to study how the choice of k affects the performance, and the results are shown in Figures 5.8–5.10. We can observe from the figures that when the number of functional regions k increases the runtime of all algorithms decreases. The reason is that for a large k it needs fewer steps to obtain a result, and the information loss decreases when k increases since fewer blocks are aggregated.

Figure 5.7: Comparing Algorithms on Singapore

Figure 5.8: Varying k on London

Figure 5.9: Varying k on Beijing

Figure 5.10: Varying k on Singapore

5.3.2.3 Varying the Size of Category Distribution d

This experiment is to study how the choice of d affects the performance, and the results are shown in Figures 5.11–5.13. It is observed from the figures that the runtime increases when the size of category distribution d increases, and the reason is that it costs $O(d)$ time to compute the KL divergence of a distribution of size d .

Figure 5.11: Varying d on London

5.3.2.4 Scalability Study

In this experiment, we study the performance when the size of the dataset varies. To conduct the experiment, we use the largest dataset London and choose different percentage of POIs to compute the functional regions. The result is shown in Figure 5.14, and we can observe that the runtime of the algorithms does not vary much when the size of

Figure 5.12: Varying d on Beijing

Figure 5.13: Varying d on Singapore

dataset varies. The reason is that the runtime of the algorithms are essentially based on the blocks, and the number of POIs does not affect the input of the algorithms.

Figure 5.14: Scalability on London

5.4 Summary

We have developed a visualized functional region exploration system for geospatial data. We define the FRS problem that partitioning the space into functional regions with minimum information loss. We show that the problem can be converted to a set partitioning problem on graph, and prove that it is NP-hard to be solved. We design a greedy algorithm that organizing the space from bottom to top, and it is guaranteed to terminate in limited steps. To improve the efficiency, we design an improved greedy algorithm that terminates in fewer steps. Finally, we use a post-processing algorithm to improve the cost of a result.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we develop several techniques to address the limitations of existing geospatial objects querying and visualized exploration problems.

In Chapter 3, we study the problem of answering *mCK* queries. We prove that this problem is NP-hard, which is not established in previous work. We propose a 2-approximation greedy approach as a baseline. Utilizing this greedy method, we first devise an approximation algorithm *SKEC* that aims at finding the smallest circle that can enclose a group of objects covering all query keywords. We prove that its approximation ratio is $\frac{2}{\sqrt{3}}$. *SKEC* has a high complexity, and we design another two approximation algorithms, *SKECa* and *SKECa+*, to find such a circle approximately for better efficiency. Their approximation ratio is $(\frac{2}{\sqrt{3}} + \epsilon)$, where ϵ can be an arbitrarily small positive value. We also design an exact algorithm utilizing *SKECa+* to reduce the exhaustive search space significantly. Extensive experiments were conducted, which verifies our theoretical analysis and shows that our exact algorithm outperforms the best known solution by an order of magnitude.

In Chapter 4, we focus on an interactive visualized exploration system for geospatial data, which took representativeness, visibility, zooming consistency, and panning consistency into consideration. We first proposed the SOS problem to select top-*k* representative objects from the current region of interest, and any two selected objects should not be too close to each other for users to distinguish in the limited space of a screen or windows. We proved that it is an NP-hard problem, and developed an approximation algorithm with performance guarantees. To provide a seamless experience for users to interactively explore the data when navigating the map, we formally defined the ISOS problem, and

proposed a pre-fetching solution on top of our greedy algorithm to significantly improve the efficiency by almost 2 orders of magnitude. We enhanced the efficiency of the proposed solutions for large datasets by a sampling technique with theoretical guarantee. Our experimental study demonstrated the efficiency and scalability of our approach. The significance of this work are three-folds: (1) This is the first work studying the SOS problem, while most previous work ignore the representative constraint except for [DP12]. As compared to [DP12] which only adopt spatial distance among objects in qualifying the representative constraint, we accept any metric as per user’s own preference and contents of objects (i.e. users can include the similarity at textual, semantic, spatial dimensions so long as the objects contain such information). (2) Our ISOS problem drops a common assumption from all previous work, i.e. the zoom levels and region cells are pre-defined and indexed, and objects are selected from such region cells at a particular zoom level rather than from user’s current region of interest (which in most cases do not correspond to the pre-defined cells). It results in extra challenge as we need to do object selection via online computation. (3) To our best knowledge, this is the first work that is able to meet all the four features aforementioned to achieve an interactive visualization map exploration system.

In Chapter 5, we developed a visualized functional region exploration system for geospatial data. We first define the FRS problem that partitioning the space into functional regions with minimum information loss. We show that the problem can be converted to a set partitioning problem on graph, and prove that it is NP-hard to be solved. We design a greedy algorithm that organizing the space from bottom to top, and it is guaranteed to terminate in $O(n)$ steps. To improve the efficiency, we design an improved greedy algorithm that terminates in $O(\log n)$ steps. Finally, we propose a post-processing technique to improve the quality of results.

6.2 Future Work

My research opens several potential directions for future work.

6.2.1 Extension of *mCK* Query

For the problem of *mCK* queries, it would be interesting to loose the hard constrain of covering exact *m* keywords. In reality, people may tend to benefit from approximate result that may only covering a part of keywords, which may involve the problem of

ranking the importance of query keywords. For example, a user issues a *mCK* query of 4 keywords, and it is possible there may be no such group of objects that can cover all of them. In this case, a result of objects covering 3 keywords is a second choice to users. However, how to rank the importance of the keywords is an issue to study in this work.

Other directions of this problem include how to solve the *mCK* query in parallel, and how to return the top-k results. Designing a distributed solution is always challenging for geospatial object queries, and such a parallel solution can provide a promising extension for a practical system. The top-k result of *mCK* query is also an interesting problem, while an issue in this problem is how to weight the importance of a group of objects.

6.2.2 Time-aware Geospatial Objects Exploration

In the further, we would like to discover other views of the geospatial objects exploration. A potential direction is exploring the geospatial objects with timestamps, i.e., the spatial temporal objects. Most of the existing exploration problems can be extended to study how the exploration evolves over time. For example, the representative set of objects in the SOS problem may change if the dataset is continuously updated. For the same reason, studying the change of functional regions over time will be an interesting view of geospatial objects exploration. It is challenging to study such problem if the geospatial objects are dynamically changed.

6.2.3 Diverse Geospatial Objects Exploration

Query result diversification, as mentioned in Section 2.3.2, is an interesting direction of studying geospatial objects. However, there are few works focus on the problem of diverse geospatial objects exploration. To explore the diversified geospatial objects, we need to know the knowledge of how the data distributes in the space. It is challenging to solve this problem in two aspects: first, it is challenging to design a function to measure the diversity of geospatial objects. Most of existing studies focus on ad-hoc solutions to a specified type of data, and measuring the diversity in a generic way is an interesting problem; second, the technique of exploring diversified geospatial objects may face the problem of efficiency, which may affect the experience of online exploration.

Appendix A

List of Publications

- **Tao Guo**, Xin Cao, Gao Cong. Efficient Algorithms for Answering the m-Closest Keywords Query. Proceedings of the ACM SIGMOD International Conference on Management of Data (**SIGMOD 2015**), pages 405-418, Melbourne, Australia, May 31 - June 4, 2015.
- Xin Cao, Gao Cong, **Tao Guo**, Christian S. Jensen, Beng Chin Ooi. Efficient Processing of Spatial Group Keyword Queries. ACM Transactions on Database Systems (**TODS**), June 2015, 40(2): 13.
- **Tao Guo**, Xin Cao, Gao Cong, Lu Jiaheng. Distributed Algorithms on Exact Personalized PageRank. Proceedings of the ACM SIGMOD International Conference on Management of Data (**SIGMOD 2017**), pages 479-494, Chicago, Illinois, USA, May 14 - 19, 2017.
- **Tao Guo**, Kaiyu Feng, Gao Cong, Zhifeng Bao. Efficient Selection of Geospatial Data on Maps for Interactive Visualized Exploration. (Submitted)
- Kaiyu Feng, **Tao Guo**, Gao Cong, Sourav Bhowmick and Shuai Ma. SURGE: Continuous Detection of Bursty Regions over a Stream of Spatial Objects. (Submitted)

References

- [AK11] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 781–792. ACM, 2011.
- [BCS16] Leilani Battle, Remco Chang, and Michael Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1363–1375, 2016.
- [BGM12] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.
- [BJ92] Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3):153–159, 1992.
- [BSC13] Leilani Battle, Michael Stonebraker, and Remco Chang. Dynamic reduction of query result sets for interactive visualizaton. In *Big Data, 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.
- [CCJO11] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.
- [CCJW13] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.
- [CHD⁺11] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

REFERENCES

- [CJW09] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
- [CMS95] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14(3):203–232, July 1995.
- [Coc07] William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.
- [CWR10] Ariel Cary, Ouri Wolfson, and Naphtali Rish. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*, pages 87–95, 2010.
- [DFHR08] I. De Felipe, V. Hristidis, and N. Rish. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.
- [DP12] Marina Drosou and Evaggelia Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment*, 6(1):13–24, 2012.
- [DP14] Marina Drosou and Evaggelia Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1102–1116, 2014.
- [DSL⁺12] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Halevy. Efficient spatial sampling of large geographical tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 193–204. ACM, 2012.
- [EH72a] D.J. Elzinga and D.W. Hearn. The minimum covering sphere problem. *Management Science*, 19(1):96–104, 1972.
- [EH72b] Jack Elzinga and Donald W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.
- [Fel08] William Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.
- [FLZ⁺12] Ju Fan, Guoliang Li, Lizhu Zhou, Shanshan Chen, and Jun Hu. SEAL: Spatio-textual similarity search. *PVLDB*, 5(9):824–835, 2012.

REFERENCES

- [FMT12] Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 421–432. ACM, 2012.
- [FT94] Andrew U Frank and Sabine Timpf. Multiple representations for cartographic objects in a multi-scale treean intelligent graphical zoom. *Computers & Graphics*, 18(6):823–829, 1994.
- [GF64] Bernard A Galler and Michael J Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.
- [HHLM07] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, page 16, 2007.
- [Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
- [HS99] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.
- [IPC15] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 277–281. ACM, 2015.
- [KA11] Mehdi Kargar and Aijun An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.
- [KBP⁺15] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *Proceedings of the VLDB Endowment*, 8(5):521–532, 2015.
- [KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

REFERENCES

- [KVSZ14] Pimin Konstantin Kefaloukos, Marcos Vaz Salles, and Martin Zachariassen. Declarative cartography: In-database map generalization of geospatial datasets. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1024–1035. IEEE, 2014.
- [LLF12] Sitong Liu, Guoliang Li, and Jianhua Feng. Star-join: spatio-textual similarity join. In *CIKM*, pages 2194–2198, 2012.
- [LLZ⁺11] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. IR-Tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.
- [LWWF13] Cheng Long, Raymond Chi-Wing Wong, Ke Wang, and Ada Wai-Chee Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700, 2013.
- [Meg82] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. In *FOCS*, pages 329–338, 1982.
- [MSV15] Mohammad Mahdian, Okke Schrijvers, and Sergei Vassilvitskii. Algorithmic cartography: Placing points of interest and ads on maps. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 755–764. ACM, 2015.
- [NAS12] Sarana Nutanong, Marco D Adelfio, and Hanan Samet. Multiresolution select-distinct queries on large geographic point sets. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 159–168. ACM, 2012.
- [PD95] Enrico Puppo and Giuliana Dettori. Towards a formal model for multiresolution spatial maps. In *International Symposium on Spatial Databases*, pages 152–169. Springer, 1995.
- [PSA14] Shangfu Peng, Hanan Samet, and Marco D Adelfio. Viewing streaming spatially-referenced data at interactive rates. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 409–412. ACM, 2014.

REFERENCES

- [QYC12] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *Proceedings of the VLDB Endowment*, 5(11):1124–1135, 2012.
- [RJN12] João B. Rocha-Junior and Kjetil Nørvrule(a)g. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179, 2012.
- [SDY⁺12] Shuo Shang, Ruogu Ding, Bo Yuan, Kexin Xie, Kai Zheng, and Panos Kalnis. User oriented trajectory search for trip recommendation. In *EDBT*, pages 156–167, 2012.
- [Ser74] Robert J Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, pages 39–48, 1974.
- [SM89] K Stuart Shea and Robert B McMaster. Cartographic generalization in a digital environment: When and how to generalize. In *Proceedings of Auto-Carto*, volume 9, pages 56–67, 1989.
- [WCLY15] Lu Wang, Robert Christensen, Feifei Li, and Ke Yi. Spatial online sampling and aggregation. *Proceedings of the VLDB Endowment*, 9(3):84–95, 2015.
- [WJT03] J Mark Ware, Christopher B Jones, and Nathan Thomas. Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science*, 17(8):743–769, 2003.
- [ZCM⁺09] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.
- [ZOT10] Dongxiang Zhang, Beng Chin Ooi, and Anthony K. H. Tung. Locating mapped resources in Web 2.0. In *ICDE*, pages 521–532, 2010.
- [ZSYY13] Kai Zheng, Shuo Shang, Nicholas Jing Yuan, and Yi Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.
- [ZTT13] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.
- [ZZZL13] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top-k spatial keyword search. In *ICDE*, pages 901–912, 2013.