Selecting the best group of objects in spatial databases and graphs

Feng, Kaiyu

2018

Feng, K. (2018). Selecting the best group of objects in spatial databases and graphs.
Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73245>

<https://doi.org/10.32657/10356/73245>

SELECTING THE BEST GROUP OF OBJECTS IN SPATIAL DATABASES AND GRAPHS

A thesis submitted
for a degree of Doctor of Philosophy
by

FENG KAIYU

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 for his patient guidance and valuable suggestions. Over the past four years, he has taught me a lot in the area of spatial database and mining. His enthusiasm in pursuing great research has always encouraged me. I also appreciate my co-supervisor Prof Jane Z. Wang and mentor Dr. Zhiqi Shen for their kind support. I also would like to thank Prof. Sourav S Bhowmick, Prof. Shuai Ma, Prof. Chunyan Miao, and Prof. Wen-Chi Peng for their kind suggestions to my research.

I would like to express my love to my girlfriend Lanting Fang for her support and encouragement. I also devote my love and gratitude to my parents.

I would like to appreciate my friends in NTU. They include Xin Cao, Quan Yuan, Tuan Anh Nguyen, Lisi Chen, Shanshan Feng, Tao Guo, Kaiqi Zhao, Zhida Chen, Daniel Rugeles Barbosa, 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, Shaokai Wang, Jin Yao Chin, Tran Dang Quang Vinh, and Yue Chen, are being or have been supervised by Dr. Gao Cong. I appreciate their accompany and support in the past four years.

Last but not least, I would like to thank the awesome facilities and technical support provided by the DMAL lab, and IGS-LILY, Nanyang Technological University.

Contents

Acknowledgments	i
List of Figures	vii
List of Tables	ix
Summary	x
1 Introduction	1
1.1 Background and Motivations	1
1.2 Problems and Research Scope	2
1.2.1 Best Aggregation Region Detection	3
1.2.2 Bursty Region Detection	4
1.2.3 Attribute-based Similar Region Detection	5
1.2.4 Influential Organizers Detection	6
1.3 Approach and Methodology	7
1.4 Summary of Contributions	9
2 Literature Survey	11
2.1 Region Detection in Spatial Data	11
2.1.1 Region Detection Queries	11
2.1.2 Other Complementary Studies	15
2.2 Social Influence and Team Formation	16
2.2.1 Social Network and Social Influence	16
2.2.2 Community Detection and Community Search	17
2.2.3 Team formation	18

3	Best Aggregation Region Detection	19
3.1	Best Region Search (BRS) Problem	19
3.2	An Exact Solution	21
3.2.1	The SIRI Problem	21
3.2.2	Disjoint Regions	23
3.2.3	Disjoint Region-based Solution	24
3.2.4	Maximal Slab-based Solution	26
3.2.5	Search Space Reduction by Slicing	29
3.2.6	The SliceBRS Algorithm	30
3.3	Approximate Solution	33
3.3.1	Overview	33
3.3.2	c -cover	33
3.3.3	Selection of c -cover	34
3.3.4	Generate a New Instance of BRS	37
3.3.5	The CoverBRS Algorithm	38
3.3.6	Approximation Ratio	39
3.4	Experimental Study	41
3.4.1	Experimental Setup	41
3.4.2	Quality and Efficiency	44
3.4.3	Usefulness of Optimization Strategies	45
3.4.4	Effect of Slice Width	48
3.4.5	Effect of the Query Rectangle	49
3.4.6	Scalability with Graph Size	50
3.4.7	Application to the MaxRS Problem	51
3.5	Summary	52
4	Bursty Region Detection	53
4.1	Problem Statement	53
4.1.1	Terminology	53
4.1.2	Burst Score	54
4.1.3	Continuous Bursty Region Detection (SURGE) Problem	55

4.2	An Exact Solution	55
4.2.1	The cSPOT Problem	56
4.2.2	Detecting Bursty Point on a Snapshot	58
4.2.3	Handling the Stream	60
4.3	Approximate Solutions	66
4.3.1	A Grid-based Solution	66
4.3.2	A Multi-Grid-Based Solution	68
4.4	Top-k Bursty Region Detection	69
4.4.1	Definition	69
4.4.2	Extension of the Exact Solution	70
4.4.3	Extension of the Approximate Solutions	72
4.5	Experimental Study	73
4.5.1	Experimental Setup	73
4.5.2	Evaluation of the Exact Solution	74
4.5.3	Evaluation of the Approximate Solutions	76
4.5.4	Effect of α	77
4.5.5	Scalability	78
4.5.6	Finding Top-k Bursty Regions	79
4.5.7	Case Study	80
4.6	Summary	81
5	Attribute-based Similar Region Detection	83
5.1	Problem Statement	83
5.1.1	Terminology	83
5.1.2	Data Distribution of an Attribute	84
5.2	A Proposed Solution	86
5.2.1	The ARI Problem	86
5.2.2	A Grid-based Algorithm	88
5.3	Experiments	93
5.3.1	Experimental Setup	94
5.3.2	Experimental Results	94
5.4	Summary	96

6	Influential Organizers Detection	98
6.1	The Influential Cover Set (ICS)	
	Problem	98
6.1.1	Independent Cascade (IC) Model	99
6.1.2	Problem Definition	99
6.2	Greedy solutions	101
6.2.1	Score-based Greedy Algorithm	101
6.2.2	PigeonGreedy Algorithm	102
6.3	Approximation Solutions	103
6.3.1	Terminology	103
6.3.2	The PICS Algorithm	105
6.3.3	Approximation Ratio Analysis of PICS	107
6.3.4	Optimized PICS Algorithm	110
6.4	Performance Study	116
6.4.1	Experimental Setup	116
6.4.2	Experimental Results	119
6.5	Summary	125
7	Conclusions and Future Work	127
7.1	Conclusions	127
7.2	Future Directions	128
	Appendix - List of Publications	130
	References	131

List of Figures

1.1	Example for the <i>BRS</i> problem.	3
1.2	Motivating example.	4
3.1	Reduction from <i>BRS</i> to <i>SIRI</i>	22
3.2	Disjoint regions.	24
3.3	$O(n^2)$ maximal regions	24
3.4	Example of maximal slabs.	25
3.5	Find maximal slabs.	25
3.6	Equal-width slices.	31
3.7	c -cover	31
3.8	Selection of c -cover T	36
3.9	Approximate Ratio.	40
3.10	Quality vs. kq	44
3.11	Runtime vs. kq	45
3.12	Quality vs. kq	46
3.13	Runtime vs. kq	47
3.14	Runtime of <i>SliceBRS</i> with and without cutting the space into slices. . . .	48
3.15	Runtime vs. slice width in Application 1.	49
3.16	Runtime vs. slice width in Application 2.	50
3.17	Effect of Aspect Ratio.	50
3.18	Runime vs. graph size.	50
4.1	Reduce to cSPOT problem	57
4.2	Illustration of bursty point detection.	60
4.3	Cell upper bound.	64

4.4	Proof for Theorem 10	68
4.5	A tight example.	68
4.6	Runtime of CCS, B-CCS, Base and aG2.	75
4.7	Runtime performance of GAPS and MGAPS.	77
4.8	Runtime performance w.r.t. α on US.	78
4.9	Scalability study.	79
4.10	Top- k bursty regions detection.	80
4.11	Bursty Region about “concert”.	82
4.12	Bursty Region about “parade”.	82
5.1	Example.	85
5.2	Reduce to ARI problem	87
5.3	Example of the Sweep line algorithm.	92
5.4	Runtime vs. rectangle size.	95
5.5	Runtime vs. grid granularity.	96
5.6	Scalability.	97
6.1	An example of the ICS problem.	103
6.2	Influence spread under influence probabilities generated by four methods.	118
6.3	Runtime vs. query groups.	120
6.4	The influence spread vs. query groups.	121
6.5	Influence spread vs. $ Q $ ($k = 5$).	123
6.6	Influence spread vs. k ($ Q = 10$).	124
6.7	Computing influence by MC simulations.	125
6.8	Runtime vs. graph size.	125
6.9	Runime of PICS and PICS+.	125

List of Tables

3.1	Table of notations	20
3.2	Summary of datasets for Application 1.	42
3.3	Summary of datasets for Application 2.	42
3.4	Effectiveness of maximal regions.	46
3.5	Effectiveness of maximal slabs.	47
3.6	Effectiveness of c -covers.	48
3.7	The ratio of <i>SliceBRS</i> 's runtime to <i>OE</i> 's runtime.	51
4.1	Table of notations	54
4.2	Datasets.	73
4.3	Ratio of rectangle messages that trigger a search vs. window size for CCS and B-CCS.	76
4.4	Approximate ratio vs. the size of window.	78
4.5	Approximate ratio vs. α	79
5.1	Example of Spatial Objects.	84
5.2	Dataset properties.	94
6.1	Table of notations.	99
6.2	Examples for UA.	112
6.3	Dataset properties.	117
6.4	Comparison with an IM technique: Jaccard similarity (JC) and attribute coverage rate (ACR).	119
6.5	Success rate of SG and PG.	121
6.6	Success rate of SG and PG (Varying $ Q $).	122
6.7	Success rate of SG and PG (Varying k).	122

Summary

With the prominence of GPS-enabled mobile devices, people can easily acquire their locations and access online services anywhere. The spatial information bridges the gap between the offline world and the online social networks, which leads to the germination of the location-based social networks (LBSNs) and event-based social networks (EBSNs). For instance, users can share their feelings and locations with their friends through LBSNs like *Foursquare*, and post a review of a restaurant through *Yelp*. As another example, people can find interesting local events to attend, or organize their own events through *Meetup*. We refer to the geo-tagged data (*e.g.*, posts, check-ins, reviews, POIs) and the users in the social networks as *objects*. The availability of large scale social data paves the way to design data-driven approaches for various applications and tasks. We are interested in one type of queries called the group optimization query. Specifically, the group optimization query aims to select a group of objects which together satisfy a specified constraint such that the pre-defined score function of the group is optimized.

In this dissertation, we define and investigate novel group optimization queries on spatial data and social networks. Specifically, for group optimization queries on spatial data, we define an axis-aligned rectangular region of a given size as the spatial constraint. Given a size of a rectangle, all the selected spatial objects should be close enough such that they can be covered by a rectangle of the given size. On the other hand, for group optimization queries in social networks, we define a set of attributes as the attribute constraint. All the selected nodes should together cover the specified set of attributes. With the spatial constraint and the attribute constraint, we define and investigate group optimization queries on spatial data and social networks by adopting different functions to measure the quality of the selected objects.

We conduct three studies on spatial data, namely *best aggregation region detection*, *bursty region detection* and *attribute-based similar region detection*. We also conduct a study on social networks, namely *influential organizers detection*.

In our study of *best aggregation region detection*, we propose the best region search (*BRS*) problem. Specifically, given a set O of spatial objects, the size $a \times b$ of a query rectangle, and a submodular monotone aggregate score function, the *BRS* problem aims at identifying a rectangular region of size $a \times b$ such that the *aggregate score* of the spatial objects inside the region is maximized. We then propose an algorithm to find the exact location of the region with the maximum score. By assuming that slight imprecision to

the solution is acceptable, we further propose an algorithm which can find an approximate answer to the *BRS* problem bounded by a constant.

In our study of *bursty region detection*, we propose a problem which utilizes such spatial object stream to detect and maintain *bursty region* of a given size in a specified geographic area in real time. We adopt two sliding windows to model the burstiness of a region. To handle spatial streams with high arrival rates, we design several pruning techniques to avoid frequent recomputation. In addition, we propose two approximate algorithms with better efficiency. We also extend the proposed solutions to support continuous detection of top- k bursty regions.

In our study of *attribute-based similar region detection*, we propose the *attribute-based similar region search* (ASRS) problem, which aims at finding a region of the same size as the query rectangle such that the attribute distance between the two regions is minimized. We propose a grid-based method to address the ASRS problem efficiently.

In our study of *influential organizers detection*, we formulate the *influential cover set* problem, which aims to select k users who can together cover a set of required skills and their influence is maximized. We adopt the independent cascade model to evaluate the influence of users. We first propose two heuristic greedy algorithms which are very efficient. The third algorithm has an approximation ratio of 2. It guarantees to find a feasible solution if any.

Chapter 1

Introduction

In this chapter, we first introduce the background and motivations of group optimization queries on spatial data and social networks in Section 1.1. Next, we present our problems and research scope in Section 1.2. We then introduce an overview of our methodology in Section 1.3. Finally, we summarize the main contributions of our studies in Section 1.4.

1.1 Background and Motivations

With the prominence of GPS-enabled mobile devices, people can easily acquire their locations and access online services anywhere. The spatial information bridges the gap between the offline world and the online social networks, which leads to the germination of the location-based social networks (LBSNs) and event-based social networks (EBSNs). For instance, users can share their feelings and locations with their friends through LBSNs like *Foursquare*, and post a review of a restaurant through *Yelp*. As another example, people can find interesting local events to attend, or organize their own events through *Meetup*. We refer to the geo-tagged data (*e.g.*, posts, check-ins, reviews, POIs) and the users in the social networks as *objects*.

With the rapid development of location-based services and event-based services, the scale of the social data is growing in an explosive rate. For example, 10 million geo-tagged tweets are generated each day in *Twitter*¹. As of August 2017, *Meetup* claimed to have 32.30 million members in 182 countries². The availability of such large scale data paves the way to design data-driven approaches for various applications and tasks. We consider the following two examples.

Example 1: George is planning a trip to Rome. Rather than visiting a region in Rome with a specific attraction (*e.g.*, shopping), he wishes to visit a region that has the most

¹<https://www.mapbox.com/blog/twitter-map-every-tweet/>

²<https://www.meetup.com/about/>

diverse collection of services and attractions (*e.g.*, cinemas, shopping malls, restaurants, and museums). This will enable him to experience many different attractions and services in one place without the need to travel to different regions to experience them all together. How can George select the “most diversified region” in Rome? $\square$

Example 2: Suppose people want to organize a social event through the event-based social network services. They want to select a group of organizers to organize an event. Intuitively, the organizers should come from different countries to guarantee the cultural diversity. Moreover, the organizers should be influential so that they can attract the as many people as possible to contribute to the event. For example, to organize a conference, we may want to choose a small group of program chairs whose locations cover Europe, Asia, and North America and are able to influence the largest number of people in the community to participate in the conference. How should we select the chairs? $\square$

The applications in the aforementioned examples naturally raise great challenges and opportunities for researchers to design novel queries. Observe the aforementioned two applications. Despite the differences in domain, there is one running theme throughout the problems encountered: we wish to find a group of objects with a certain constraint such that the score of the group is optimized. Specifically, in the first example, we wish to find a group of POIs with a spatial constraint (the POIs should be close enough to be located inside a region) such that the “diversity” of the objects is maximized. On the other hand, in the second example, we wish to find a group of users with an attribute constraint (the users should together have the required skills) such that the “influence” of the organizers is maximized. We generalize such kind of queries and refer to them as the *group optimization query*.

In this dissertation, we define and investigate novel group optimization queries on spatial data and social networks. Specifically, for group optimization queries on spatial data, we define an axis-aligned rectangular region of a given size as the spatial constraint. Given a size of a rectangle, all the selected spatial objects should be close enough such that they can be covered by a rectangle of the given size. On the other hand, for group optimization queries in social networks, we define a set of attributes as the attribute constraint. All the selected nodes should together cover the specified set of attributes. With the spatial constraint and the attribute constraint, we define and investigate group optimization queries on spatial data and social networks by adopting different functions to measure the quality of the selected objects.

1.2 Problems and Research Scope

My research aims at investigating group optimization queries on spatial data and social networks. Thereby, four problems are defined and investigated in this dissertation.

 Figure 1.1: Example for the *BRS* problem.

Specifically, we study three tasks on spatial data – (1) Best aggregation region detection; (2) Bursty region detection; and (3) Attribute-based similar region detection. In addition, we study a problem in social networks – (4) Influential organizers detection. For each of the tasks, we formulate a problem and propose efficient and effective approaches to address it.

1.2.1 Best Aggregation Region Detection

In our first study, we propose a new problem called the *best region detection* problem. Specifically, given a set O of spatial objects, the size $a \times b$ of a query rectangle, and a submodular monotone aggregate score function, the *BRS* problem aims at identifying a region of size $a \times b$ such that the *aggregate score* of the spatial objects inside the region is maximized.

In the *BRS* problem, there is an infinite number of points in the space and it is prohibitively expensive to consider all these points in the search of the best region. This is more so because submodular monotone functions are often expensive to compute [KKT03a]. Furthermore, a user may search for the best region in an *exploratory* manner. That is, she may initiate a search with a specific query rectangle as input, view the corresponding results, iteratively refine the query rectangle (by increasing or decreasing a or b) and execute the refined search until she is satisfied with the search results. Such exploratory framework demands techniques that can efficiently process the *BRS* queries over very large volumes of spatial objects.

At first glance, it may seem that the *BRS* problem can be addressed by the techniques designed to tackle the *maximizing range sum (MaxRS)* problem [IA83, NB95, CCT12]. The *MaxRS* problem takes as input a set O of weighted spatial objects and the size $a \times b$ of a query rectangle. The problem aims at finding the rectangular region of $a \times b$ size such that the sum of weights of the spatial objects inside the region is maximized. However, this is not the case. Consider the example shown in Figure 1.1, the dashed-line rectangular region is the result for the *MaxRS* problem since there are four spatial objects in the region. However, the solid-line rectangular region is the answer to our *BRS* problem. Instead of the SUM function, the *BRS* problem is a generalized problem that can employ *any* submodular monotone function. Thus, the *MaxRS* problem is a special case of the *BRS* problem.

Figure 1.2: Motivating example.

1.2.2 Bursty Region Detection

In the second study, we formulate a novel *continuous bursty region detection* (SURGE) problem motivated by the following examples.

Example 3: The world regularly faces the challenge of tackling a variety of virus epidemics such as SARS, MERS, Dengue, and Ebola. Most recently, the outbreak of mosquito-borne Zika virus started in Brazil in 2015. Hence, the Center for Disease Control and Prevention needs to continuously monitor different areas for possible Zika outbreak and issue alerts to people who are traveling to or living in regions affected by Zika. Since early detection of such outbreak is paramount, how can we identify potential Zika-affected region(s) in real time?

One strategy to address this issue is to continuously monitor geo-tagged tweets (i.e., spatial objects) coming out of a specific area (e.g., Florida) and detect regions where there are sudden bursts in tweets related to Zika (e.g., containing Zika-related keywords) in real time. Observe that these “bursty regions” are dynamic in nature. However, it is computationally challenging to continuously monitor massive streams of spatial objects and detect bursty regions in real time. $\square$

Example 4: Online transportation network companies such as *Uber*, *Lyft*, and *Didi Dache* have disrupted the traditional transportation model and have gained tremendous popularity among consumers³. Consumers can submit a trip request through their mobile apps. If a nearby driver accepts the request, he will pickup the consumer.

Although this disruptive model has benefited many drivers and consumers, the latter may have to wait for a long time for a car when the number of car requests significantly surpasses the supply of nearby drivers. Clearly, it is beneficial to both passengers and

³In 2017, Uber is available in over 81 countries and 570 cities worldwide.

drivers if we can notify idle drivers in real time whenever there is a sudden burst in consumer demand in areas of interest to them. An additional benefit to the drivers is that the trip fare may be increased due to the “surge pricing” policy⁴ where the companies may increase a trip price significantly when demand is high. For instance, consider Figure 1.2, which shows the trip requests in two time windows $[t_1, t_2]$ and $[t_2, t_3]$. Suppose a driver is only interested in the area shown by dashed rectangle to pick up passengers. Observe that there is a burst of trip requests in regions r_1 and r_2 (both increased by 3). If the app can notify the driver in real time about these two regions, then he can move in there to pickup potential passengers. Note that such soaring demand is not always predictable as it may not only occur during holidays or periodic events (*e.g.*, new year’s eve) but also due to unpredictable events such as subway disruption, concerts, road accident, inclement weather, and terrorist attack. $\square$

There are two common themes in the two examples. First, we need to continuously monitor a large volume of spatial objects (*e.g.*, trip requests and geo-tagged tweets) to detect in real time one or more regions that show relatively large spike in the number of spatial objects (i.e., bursty region) in a given time window. Second, a user needs to specify as input the size $a \times b$ of rectangular-shaped bursty region that one wishes to detect. For instance, in Example 4 different drivers may prefer bursty regions of different sizes according to their convenience.

Specifically, given a region size $a \times b$ and an area A , the aim of the SURGE problem is to continuously detect a region of the specified size in A that demonstrates the *maximum burstiness* from a stream of spatial objects. We define a general function based on the sliding window model to model a region’s burstiness. We also extend our SURGE problem to detect *top-k* bursty regions as in certain applications one may be interested in a list of such regions.

The SURGE problem and its *top-k* variant are challenging as we need to handle rapidly arriving spatial objects in high volumes to efficiently detect and maintain bursty regions. For example, 10 million geo-tagged tweets are generated each day in *Twitter*⁵.

1.2.3 Attribute-based Similar Region Detection

In our third study, we propose a novel problem called the *attribute-based similar region search* (ASRS) problem. Consider a set of spatial objects, each of which is associated with a set of attributes. Given a query region r_q of size $a \times b$, a preferred attribute A , the ASRS problem aims at finding a region of the size $a \times b$, such that the data distribution of the attribute A of the region is most similar to the query region.

⁴For example, the price increased 10X on new year’s eve in 2016 in the United States (www.geekwire.com/2016/customers-complain-uber-prices-surge-near-10x-new-years-eve/)

⁵<https://www.mapbox.com/blog/twitter-map-every-tweet/>

The availability of geo-tagged objects can facilitate spatial analysis. In reality, the spatial objects tend to distribute heterogeneously, i.e., different regions have distinct characteristics according to the utilities of objects in the area [TK97]. In data exploration, a fundamental task is to understand the distribution of spatial objects along a specified attribute in a region, where the attribute can be *category*, *ratings*, *hashtags*, etc. Such distribution reveals significant properties of the region. For example, consider a set of POIs. In order to get an overview of the functionality of a region, people may want to know the distribution of spatial object along the *category* attribute. If many of the spatial objects in the region are categorized to be *shopping malls* and *restaurants*, this region is highly likely to be a commercial area. If many of the spatial objects inside the region are *apartments*, then the region is possibly a residential area.

A natural question then is how to find the region that best matches the user's demand. Since it is unfriendly for the users to describe every components of his/her desired region, a simple way is to ask the user to provide an example region as query and the most similar region is returned to the user as the result. Intuitively, two regions with similar distributions of an attribute usually share common properties. Such detection of similar region is useful in various applications. The owner of a successful coffee shop may want to extend the business to a new city. He can search for a region in the new city that best matches the surroundings of the current coffee shop, which may attract customers in same consumption level, e.g. college students or office workers. When a tourist uses the map services to browse a city, he may find some regions are interesting to explore. Meanwhile, some other similar attracting regions can be detected and recommended for expanding his choices.

It is challenging to solve the ASRS problem efficiently. The reason is that there are $O(n^2)$ possibilities to locate the region, where n is the size of the dataset. The straightforward way to solve this problem would cause $O(n^2 \cdot n_c \cdot m)$ time, where n_c is the average objects in a region and m is the cardinality of the attribute. In addition, different from the sum function in the *MaxRS* problem, the attribute distance in our problem is not monotonic. Specifically, let T_q be the query attribute distribution. Let O_r be a set of spatial objects. By adding a new object o into O_r , the attribute distance between O_r and O_q could either increase or decrease. As a result, the techniques designed for the *MaxRS* problem cannot be used to solve the ASRS problem.

1.2.4 Influential Organizers Detection

In our fourth study, we propose a novel *influential cover set* problem to select influential organizers on social networks. Formally, given a social network G , let each node v in G refers to a user and is associated with a set of attributes $\mathcal{A}(v)$, where an attribute may refer to an expertise topic of a user. The ICS problem is to identify a set $\{v_1, \dots, v_k\}$ of

k users in G such that (1) $\bigcup_{i=1}^k \mathcal{A}(v_i)$ covers a set of query attributes, denoted by Q , and (2) the set of k users can influence the largest number of other users in G .

At first glance, it may seem that the ICS problem can be addressed by adopting any technique designed to address the traditional influence maximization (IM) problem [KKT03b]. The influence maximization problem aims at finding a set of k users from a social network such that the influence of the k users is maximized. Unfortunately, this is not the case as state-of-the-art IM techniques do not meet the attribute covering constraint of the ICS problem. That is, in contrast to the traditional IM problem, the challenge for solving the ICS problem is to ensure that the k selected nodes cover the query attributes while their influence spread is maximized. Furthermore, in our problem setting k is significantly smaller (typically less than 6) compared to the IM problem since, as mentioned above, the number of event organizers of a particular social event is typically small. Consequently, it is imperative to design efficient techniques that leverage this unique characteristics of the seed set size.

1.3 Approach and Methodology

Best Aggregation Region Detection

To address the first problem and tackle the challenges mentioned in Section 1.2, we propose two novel algorithms called the *SliceBRS* algorithm and the *CoverBRS* algorithm. The first *SliceBRS* algorithm finds an exact answer to the *BRS* problem by reducing it to the submodular weighted rectangle intersection (*SIRI*) problem, which greatly reduces the search space. Specifically, we prove that only $O(n^2)$ candidate regions need to be considered. However, it is still inefficient to consider all $O(n^2)$ candidate regions (especially in an exploratory search environment) when the number of spatial objects is very large. This leaves us a challenge whether we can further prune the search space. The second algorithm, *CoverBRS*, assumes that slight imprecision to the solution is acceptable and finds an approximate answer to the *BRS* problem bounded by a constant. In *SliceBRS*, we propose several new concepts, including “maximal regions” and “maximal slabs” to prune the search space. Based on these concepts, we cut the space into slices and find maximal slabs in all slices. Then we prune unnecessary slices and maximal slabs to reduce the search space. It takes $O(n \times n_s)$ time to find the best point, where n is the number of spatial objects and n_s is the number of maximal slabs that we actually processed. According to our experiments, n_s is usually a very small number, which is much smaller than n . The *CoverBRS* algorithm first selects a smaller set of spatial objects T using a novel concept of *c-cover*. Then it generates a new instance of the *BRS* problem by defining a new aggregate score function and a new query rectangle on the set of spatial objects T . Lastly, it invokes the *SliceBRS* algorithm to answer

the new instance. We prove that the answer to the new instance is a constant-bounded approximate answer to the original instance. The approximate ratio is determined by the parameter c .

Bursty Region Detection

For the second research problem, we first propose an exact solution called *cell*-CSPOT to keep track of the bursty region over sliding windows. Specifically, we first reduce the SURGE problem to *continuous bursty point detection* (CSPOT) problem. Then we propose a cell-based algorithm to continuously detect the bursty point. In particular, a grid is imposed over the space. For each cell in the grid, we dynamically maintain an upper bound of the burstiness of the cell to determine when we should search the cell for the bursty region. It takes $O(|c_{max}|^2 + \log n)$ time to process a new arriving spatial object on average, where $|c_{max}|$ is the maximum number of objects that we search inside a cell, and n is the number of indexed rectangle objects.

Although *cell*-CSPOT can address the SURGE problem efficiently in several scenarios, it becomes inefficient as $|c_{max}|$ increases (*e.g.*, the sliding windows get larger, the region size gets larger, or the arrival rate of the spatial objects increases). To address this we further propose two approximate solutions, namely GAP-SURGE and MGAP-SURGE, with an $O(\log n)$ time complexity to process a spatial object. The approximation ratio is bounded by $\frac{1-\alpha}{4}$, where $\alpha \in [0, 1)$ is a parameter used in the *burst score function*.

Attribute-based Similar Region Detection In order to address the ASRS problem, we first reduce the ASRS problem to the *attribute-based rectangle intersection* (ARI) problem. We show that the result point of the reduced ARI problem is the top-right corner of the most similar region in the ASRS problem. In addition, to reduce the search space, we propose a grid-based algorithm. Specifically, for each cell in the grid, we maintain a lower bound of the attribute distance. We can greatly reduce the search space by pruning the cells whose lower bounds are larger than the current result. Finally, we present a sweep-line based algorithm to search a cell to location the region with minimum attribute distance.

Influential Organizers Detection

For the fourth research problem, we first propose two greedy solutions, namely **ScoreGreedy** and **PigeonGreedy**. Given an influential cover set query comprising query attributes Q and size k , **ScoreGreedy** selects k seed nodes in a greedy manner by considering both the number of newly covered attributes and marginal influence increase incurred by a candidate node. The **PigeonGreedy** algorithm leverages the pigeonhole principle since to cover $|Q|$ attributes with k nodes, we know at least one node in the selected seed set should cover more than $\frac{|Q|}{k}$ attributes. Based on this observation, it chooses the one with the maximum marginal influence increase among the nodes that cover at least $\frac{|Q|}{k}$ attributes.

While the greedy algorithms are efficient, they may not be able to return a seed set that covers all attributes in the query even if such a set exists. This led us to the design of the approximation algorithm **PICS** (**P**artition-based **I**nfluential **C**over **S**et) algorithm, which always returns a seed set that covers all query attributes if such a set exists, and, even better, it is an approximation algorithm with performance guarantee. Specifically, for each **partition** of the given attributes Q , we find a seed set of size k in which the attributes of each node cover a subset of the **partition** and they have the largest influence spread. Finally, it returns the seed set with the maximum influence spread. We prove that the influence of the returned seed set is bounded by an approximation ratio of 2.

Despite the benefits of **PICS** to address the ICS problem, a key limitation is that the number of **partitions** can be very large when $|Q|$ is large (*e.g.*, 115,975 at $|Q|=10$), making it computationally expensive. To improve the efficiency, we propose the notion of **cover-group** to group **partitions**. We develop an algorithm called **PICS+** for pruning the **partitions** in each **cover-group** from consideration, which cannot generate the result seed set. We show that this algorithm is instance optimal in pruning unnecessary **partitions**.

1.4 Summary of Contributions

Firstly, we introduce the best region search (*BRS*) problem that aims to find a rectangular region with a given size such that the aggregation of the spatial objects in the region is maximized. This problem is fundamental to supporting several location-based applications such as most influential region search and most diversified region search. To this end, we propose a novel algorithm called *SliceBRS* that employs several pruning strategies to find the exact answer to the *BRS* problem. Since slight imprecision is acceptable in many real-world applications, we further propose the *CoverBRS* algorithm to find a constant-bounded answer to the *BRS* problem with a much better efficiency. The experimental study demonstrates that *SliceBRS* can find the exact answer to the *BRS* problem whereas *CoverBRS* can find a competitive answer with a better efficiency. These algorithms scale well with the number of spatial objects.

Secondly, we have studied a new problem called the *continuous bursty region detection* (*SURGE*) problem to continuously detect the bursty region in a given area in real time. The *SURGE* problem is important as it can underpin various applications such as disease outbreak detection. We have proposed an exact solution and two approximate solutions for the *SURGE* problem. We have also extended these solutions to find top- k bursty regions. Finally, our experiment study with real-world datasets has demonstrated the efficiency of our framework. As part of future work, we intend to explore the *SURGE* problem in the context of road network.

Thirdly, we propose a novel problem called the *attribute-based similar region search* (*ASRS*) problem, which aims at finding a region with similar data distribution of a given

attribute to the query region. To this end, we propose an exact algorithm called *GARI* which employs a grid-based strategy to prune unnecessary cells. We also propose a sweep line based algorithm to search a cell to locate the point with the minimum attribute distance. We conduct experimental study to demonstrate the performance of the proposed algorithm. We show that the *GARI* algorithm can efficiently find the region with similar attribute distribution. We also show that the algorithm scales well *w.r.t.* the number of spatial objects.

Finally, we introduce the problem of discovering influential event organizers from an online social network. We formulate it as the problem of mining influential cover set (ICS), which is intractable. Hence, we propose two greedy algorithms, **ScoreGreedy** and **PigeonGreedy**, to find approximate solutions to the problem. Although they are efficient, they have no guarantee to find a feasible solution. This led us to propose an approximation algorithm called **PICS+** that guarantees to find a feasible solution if any. The experimental results on real-world datasets demonstrate that **PICS+** guarantees to find a feasible solution if any within reasonable time. Although the two greedy algorithms run faster, they fail to find feasible solution for a significant portion of queries.

Chapter 2

Literature Survey

My research is about group optimization queries on spatial data and social networks, which aims to identify a group of objects with spatial constraints (in spatial data) or attribute constraints (in social network) such that a specific score function of the identified objects is optimized. My four proposals can be classified into two categories: region detection and organizers detection. Therefore, in this chapter, we first survey studies related to region detection and organizers detection in Section 2.1 and 2.2. After that, we review the studies related to the social influence and burstiness, which are the score functions that are used in our proposals.

2.1 Region Detection in Spatial Data

There is a class of studies about region detection in spatial data, which is closely related to our work on region detection in spatial data. We classify the studies into two categories. The first category of studies consists of various region detection queries. In the second category of studies, we review some complementary studies for the region detection queries.

2.1.1 Region Detection Queries

We review four types of region detection queries, namely location selection query, maximum range sum query, and dense region query.

2.1.1.1 Location Selection Query

The first type of the studies is *location selection* problem. Given a set of sites and a set of objects, the location selection problem aims to find the location of a new site such that the total cost is minimized. The existing work on location selection defines different optimization objectives, based on which we classify the existing work. First, Zhang *et*

al. propose the *min-dist optimal-location query* [ZDXT06], which takes as input a set of servers, a set of clients, and a spatial region Q . The query returns a location in Q for a new server such that the average distance from each client to its closest server is minimized. The problem is also considered in the context of road networks [XYL11, CLW⁺14] with road network distance. Second, given a set of servers and a set of clients, several efforts [DZX05, WÖY⁺09] aim to find a location where a new server can maximize its influence. Here the concept of influence of a server is based on reverse nearest neighbor (RNN) [KM00] and is measured by the number of clients that are closer to it than to any other servers. Zhou *et al.* [ZWL⁺11] and Yan *et al.* [YWN11] consider the problem with extensions to the definition of influence respectively. Zhou *et al.* [ZWL⁺11] extend the definition of *influence* of a server, which is measured by the number of clients who take the server as one of their k nearest server locations. Yan *et al.* [YWN11] propose another extension to the definition of influence by allowing a server to influence more nearby clients. In addition, the problem of selecting top- k locations is also studied in the literature [XZKD05, HWQ⁺11, ZZL12].

2.1.1.2 Region Search Query

The second type of the studies is *maximum range sum* (*MaxRS*) problem. The *MaxRS* problem takes as input a set O of weighted spatial objects and the size $a \times b$ of a query rectangle. The problem aims at finding the rectangular region of $a \times b$ size such that the sum of weights of the spatial objects inside the region is maximized. Imai *et al.* propose an $O(n \log n)$ optimal algorithm [IA83] for finding the position of a rectangle of the given size enclosing the maximum number of spatial objects, where n is the number of spatial objects. This algorithm can be employed to solve the *MaxRS* problem. Later Nandy and Bhattacharya [NB95] propose another line-sweeping-based algorithm with the same $O(n \log n)$ cost. The *MaxRS* problem is systematically investigated by Choi *et al.* [CCT12] where an elegant external memory algorithm based on the in-memory $O(n \log n)$ algorithm [NB95] is proposed. Tao *et al.* [THCC13] extend the *MaxRS* problem to $(1-\epsilon)$ -approximate *MaxRS* problem and propose efficient algorithms for answering the *MaxRS* query approximately. Liu *et al.* [LYS11] study the problem of finding subject oriented top- k hot regions, which can be considered as a top- k version of the *MaxRS* problem.

The *MaxRS* problem is mostly germine to our work on best aggregate region detection problem. However, it is still a different problem for the following reasons. The *MaxRS* problem takes SUM as the aggregate score function, while our work on best aggregate region detection problem uses a general submodular monotone function as the aggregate score function. In other words, the *MaxRS* problem is a special case of our problem when the aggregate function is SUM. In addition, techniques [IA83, NB95, CCT12] developed

for the *MaxRS* problem cannot be deployed or adapted to solve our problem because their pruning techniques are tightly integrated to the SUM function and hence cannot be generalized to other submodular monotone functions.

Different from the original *MaxRS* problem, Amagata et al. [AH16] investigate the *MaxRS* problem over spatial data streams. Specifically, given a stream of weighted spatial objects, the continuous *MaxRS* problem aims to monitor the location of a rectangle of a size $a \times b$ such that the sum of the weights of the objects covered by the rectangle is maximized. In the proposed algorithm, a grid is imposed over the space, whose granularity is independent from the size of the query rectangle. For each spatial object in the stream, they generate a rectangle of a size $a \times b$ whose center is located at the spatial object. The generated rectangle is mapped to the cells with which it overlaps. For each cell, they maintain a graph where each node in the graph is a rectangle mapped to this cell, and two nodes are connected by a directed edge if they overlap with each other. The graph is used to handle the updates of the stream. For each rectangle in the cell, they maintain an upper bound to determine when to invoke the sweep-line algorithm [NB95] to find the most overlapped region inside the rectangle. With the maintained upper bounds, they use a branch-and-bound algorithm to reduce the search space. Hussain et al. [HTIA17] investigates the *MaxRS* problem on the trajectories of moving objects. Given the trajectories of a set of moving points, they aim to maintain the result of the *MaxRS* problem at any time instant. Its problem setting is different from our work on bursty region detection: it takes as input the trajectories of a set of fixed number moving objects, while in our problem, the number of spatial objects in the sliding windows may vary with time and the positions of the new arrived objects are unknown a priori.

The continuous *MaxRS* problem is closely related to our work on bursty region detection problem. The difference between the two problems is that the burst score in our problem is defined over two consecutive sliding windows, and spatial objects in different windows contribute differently to the burst score. Moreover, one issue of this solution is that they need to maintain a graph for each cell with a space cost of $O(n^2)$, where n is the number of rectangle objects that are mapped to the cell. When the number of objects mapped to a cell is large, the space cost could be extremely high. In our work on bursty region detection problem, we propose solutions which outperform their solution.

2.1.1.3 Dense Region Query

The third type of the studies is *dense region* query [JLOZ06, NR07]. Given a set of moving objects, whose positions are modeled as linear functions in Euclidean space, the dense region search problem aims to find all dense regions at query time t . Jensen et al. [JLOZ06] constraint dense regions to be non-overlapping square-shaped regions of given size, whose density is larger than a user-specified threshold. With this constraint,

they only report one set of non-overlapping dense regions. However, the non-overlapping constraint could lead to totally different results under different searching strategies for the same query. To address this issue, Ni et al. [NR07] propose a new definition of dense regions, which may have arbitrary shape and size. In the dense region search problem, the positions of the moving objects are modeled as linear functions. In addition, they have a different density function requiring different techniques to compute the score of a given region from our region detection problems.

2.1.1.4 Burst Detection

Finally, the fourth type of the studies is *burst detection*. A host of work has been done to detect temporal bursts [Kle03, ZS03, FYYL05, BS05, WZHS07]. A collection of proposals focus on detecting bursty features (represented by probability distribution of words)[Kle03, FYYL05, WZHS07]. The other work focuses on detecting a timespan over the stream such that its aggregate is larger than a threshold [ZS03, BS05]. All these burst detection problems are different from our problem of bursty region detection as they disregard the spatial information when detecting the temporal bursts.

Most germane to our problem of bursty region detection are efforts on exploring spatial-temporal bursts [MBK10, LVGT12, ZZY⁺16] albeit from different aspects. Mathioudakis et al. [MBK10] study the problem of identifying notable spatial burst out of a collection of user generated information. They divide the space into cells, and recognize two states for each cell, namely “bursty” and “non-bursty”. Characterization of cells as bursty or non-bursty is formulated as a minimization problem of an additive cost function. It differs from our problem in two key aspects. First, the spatial burst is identified as a cell in the grid whereas the bursty region in our problem can be located at any position. Second, the solution developed in [MBK10] is designed for data stored in a data warehouse, and it cannot be deployed or adapted to solve our problem, i.e., continuously detecting the spatial burst in a stream in real time.

Given a set of text streams, each associated with a geographical location, Lappas et al. [LVGT12] study the problem of identifying a combination of a temporal interval and a geographical region with unusual high frequency for a term from a set of geo-tagged text streams. Its problem setting is different from ours: Lappas et al. [LVGT12] takes as input a set of text streams with fixed geographical locations, while in our problem, spatial objects arrive as a stream and an object can be located in any location of the given space. In addition, the proposed solution can only handle a small number of text streams (tens to hundreds) due to its high computational complexity.

Given a geo-tagged tweet stream, Zhang et al. [ZZY⁺16] aim to continuously detect real-time local event. Specifically, a local event is defined as a cluster of tweets that are semantically coherent and geographically close. For each keyword in a tweet, its

burstiness is a linear combination of its temporal burstiness and its spatial burstiness with a balance parameter η . The spatiotemporal burstiness of a cluster of tweets is the aggregation of the burstiness of all the keywords in the cluster. Our problem differs from it in the following aspects. First, the bursty event is identified as a cluster of geo-tagged tweets, while our problem aims to detecting a spatial region. Second, the proposed framework is built over geo-textual stream. The textual content serves as an important feature in their system. Our problem is applicable to any kind of spatial stream.

2.1.2 Other Complementary Studies

2.1.2.1 Range Aggregate Query.

The first type of the complementary studies is *range aggregate* (RA) query. Given a set O of spatial objects and a query range q , the RA query [JL98, LM01, PKZT01, CC07, ST11] returns the total weight of the spatial objects that are inside the given query range. Most of the studies related to range aggregate query processing focus on proposing new spatial index structures to process these queries efficiently. Though the range aggregate query does not An RA query aim to answer “what is the total weight of objects in a given query region?” In contrast, our region detection problems focuses on providing answer to “where is the location of the region of a given size such that the score function of the region is optimized?” Moreover, the aggregate function used in these studies is different from the functions used in our problems.

2.1.2.2 Spatial Outlier Detection

The second type of the complementary studies is *spatial outlier detection* problem [LCK03, ZLK03, KLC06]. Lu et al. [LCK03, KLC06] investigate the spatial outlier detection problem over point data. Specifically, given a set of weighted spatial points, the spatial outlier detection problem aims to identify top m points such that their weight is greatly different from the average weight of its k nearest neighbors. Zhao et al.[ZLK03] further investigate region outliers detection over meteorological data. All these aforementioned problems focus on static data. Moreover, the outliers are selected from the data points in the spatial outlier detection problem. In contrast, in our region detection problems, the location of the region can be located at any position in the space. In addition, the spatial outlier detection problems use a totally different function to evaluate how much a data point is different from its neighbors.

2.1.2.3 Spatial Stream Management

Our problem of bursty region detection is related to spatial stream management. There has been a long stream of work on various aspects of data streams since the last decade.

Some examples are stream clustering [AHWY03, OMM⁺02], stream join processing [DGR03], and stream summarization [CM05]. Since most of these studies focus on general data streams, we only review the work that involves spatial information. Given a stream of spatial-textual objects, [WZZ⁺14] aims to estimate the cardinality of a spatial keyword query on objects seen so far. A host of work has also been done to study content-based publish/subscribe systems [WZZ⁺16, CCCT15, HLL⁺15, LWWF13, CCC13, WZZ⁺15] over spatial object streams. In these systems, streaming published items are delivered to the users with matching interests. However, none of these studies consider the problem of bursty region detection.

2.2 Social Influence and Team Formation

2.2.1 Social Network and Social Influence

2.2.1.1 Event-based Social Network

Liu *et al.* [LHT⁺12] introduce the notion of event-based social network (EBSN) which comprises online and offline social interactions. Specifically, they analyze various characteristics of EBSN such as network properties, community structures and information flow over this network. They also propose a technique for event recommendation. Our work is orthogonal to this effort as we focus on the online component of the EBSN and present an efficient and scalable technique to find influential event organizers.

2.2.1.2 Social Influence Propagation

There exists a lot of work on the problem of influence maximization in social networks. The influence maximization problem is a discrete optimization problem [KKT03b], which aims at selecting a seed set consisting of k users from the social network such that the influence of the seed set is maximized. The problem is proved to be NP-hard. Kempe [KKT03b] propose a greedy algorithm to approximately solve the problem. Specifically, the greedy algorithm add the user with the maximum marginal influence increase to the seed set iteratively until the seed set has k users. However, the greedy algorithm framework does not consider the attribute coverage when greedily selecting nodes into the seed set. Hence, the solution returned by such a framework may not satisfy the attribute coverage requirement in our problem setting.

It is #P-hard to compute the influence spread of a given seed set [CWY09]. In order to estimate the influence spread, researchers have proposed various approaches. Kempe *et al.* [KKT03b] propose a simulation-based method. Specifically, by following the propagation model, they simulate the influence propagation from the seed set for a large number of times. The average number of influenced users in the simulation results

as the estimated influence spread. As this method requires a large number of simulations, it is very computationally expensive. To mitigate this problem, Leskovec *et al.* [LKG⁺07] propose a mechanism called CELF. Instead of computing the marginal influence increase for all users in each iteration, the CELF approach only compute the marginal increase for the users who may have the largest marginal increase. Chen *et al.* propose two methods to approximate the marginal influence increase. Specifically, the first method, namely DegreeDiscount [CWY09], estimates the marginal increase of a node by its degree. The second method, namely PMIA [CWY09], propose *local influence arborescences* based on the most probable influence path between two nodes. Jung *et al.* [JHC12a] estimate the influence spread using an iterative approach, which is derived from the belief propagation approach. The concept of influence spread path is proposed to estimate the influence spread of a seed set [KKY13, LCXZ12].

2.2.2 Community Detection and Community Search

In a social network, communities are defined as non-overlapping graph partitions. The preferred structure of a community is highly dependent on the target application. Thus, various community quality scores have been defined and investigated. In this section, we classify the studies related to community into two categories.

2.2.2.1 Community Detection

Given a social network, the community detection problem aims to find all communities such that the nodes in a community are closely connected and tend to have more interactions inside the community. Newman *et al.* [NG04] propose algorithms to divide the graph into densely connected subgraphs by adopting betweenness as the measure function. Fortunato [For10] defines intra-cluster density and inter-cluster density to measure the quality of communities. Recent work investigate the community detection problem on attributed graphs [ZCY09, RFP13]. These studies consider the attribute similarity when clustering the graph. A host of work utilizes topic models to capture the contents associated to the nodes and edges [LYM14, LNMG09, NAXC08, SCFS12].

2.2.2.2 Community Search

Different from the community detection problem, the community search problem aims to find communities that can answer a query [HLX17]. For instance, a class of work studies the problem of local community search problem, which aims to find the community near the query nodes [SHK⁺10, TBG⁺13, SG10, ZHX⁺17, HL17]. Zhu *et al.* [ZHX⁺17] investigate the community search on location-based social network by proposing a new type of query named the geo-social group queries with minimum acquaintance constraints

(GSGQ). Specifically, given a query node v , a spatial constraint Λ , and a minimum degree c , the GSGQ query aims at finding a maximal result set of nodes containing the query node v such that the result set satisfies the spatial constraint and the reduced subgraph is a c -core. Fang et al. [FCLH16] propose a model for the community search problem over attributed graphs based on k -cores. Given a query node q and a minimum degree k , and a set of keywords S , the attributed community search problem aims to find all subgraphs containing the query node q such that each subgraph satisfies the structure cohesiveness (the minimum degree of each node in the subgraph is at least k) and the keyword cohesiveness (the size of the common keywords among the nodes and S is maximal). Huang et al. [HL17] further propose a k -trusses based model, which have denser structures than k -cores and support multiple query nodes.

2.2.3 Team formation

Lappas *et al.* [LLT09] consider the team formation problem in the presence of a social network of individuals. They aim to find a team with the set of required skills and minimum communication cost. The problem formulation is also adopted by some subsequent work on team formation, *e.g.*, [MDN12, ABC⁺12]. These techniques aim to form a team to implement a task, but not organize an event; and more importantly they do not consider the influence of users in team formulation. However, influential organizers are very important for the success of online event planning [RWR13].

Chapter 3

Best Aggregation Region Detection

This chapter proposes a novel problem called the best region search (*BRS*) problem. Specifically, given a set O of spatial objects, a submodular monotone aggregate score function, and the size $a \times b$ of a query rectangle, the *BRS* problem aims to find $a \times b$ rectangular region such that the *aggregate score* of the spatial objects inside the region is maximized. Two novel algorithms are proposed to tackle the *BRS* problem. The first algorithm, *SliceBRS*, finds an exact answer to the *BRS* problem. The second algorithm, *CoverBRS*, assumes that slight imprecision to the solution is acceptable and finds an approximate answer to the *BRS* problem bounded by a constant. An experimental study with real-world datasets and applications demonstrates the effectiveness and superiority of the proposed algorithms.

This chapter is organized as follows. Section 3.1 formally introduces the best region search problem. In Section 3.2, we present the exact algorithm, namely *SliceBRS*. Section 3.3 presents the approximate algorithm *CoverBRS*. The experimental study is discussed in Section 3.4. Finally, the last section summarizes this chapter. The key notations used in this chapter are given in Table 3.1.

3.1 Best Region Search (BRS) Problem

In this section, we formally introduce the *best region search* (*BRS*) problem that we address in this chapter. We consider a set of spatial objects O in a 2-dimensional space, denoted by P . Each object $o \in O$ has a location represented by $(o.x, o.y)$ in space P . We use $r_p^{a,b}$ to denote the $a \times b$ rectangular region centered at the point $p \in P$. Without ambiguity, we also use $r_o^{a,b}$ to denote the $a \times b$ rectangular region centered at a spatial object $o \in O$. For a rectangular region $r_p^{a,b}$, we denote by $O_{r_p^{a,b}}$ the set of spatial objects in O that are located in $r_p^{a,b}$. We define submodular monotone function f as follows:

Definition 1: Submodular monotone function[CR04]. A set function $f : 2^{|O|} \rightarrow \mathbb{R}$, which maps subsets of O to a real number is a monotone submodular function if for every

Table 3.1: Table of notations

Notation	Definition
P	The 2-dimensional space
O, T	The set of spatial objects
o, p	A spatial object, a point
$a \times b$	The size of the query rectangle
$r_p^{a,b}$	The $a \times b$ rectangular region centered at p
$O_{r_p^{a,b}}$	The set of objects inside the rectangle $r_p^{a,b}$
f, f_T	The submodular monotone function on O and T
R	The set of rectangles
$\mathbf{r}$	A disjoint region
S, s	A set of maximal slabs, a maximal slab

$O_i \subseteq O_j \subseteq O$ and $o \in O \setminus O_j$, it satisfies: (1) $f(O_i \cup \{o\}) - f(O_i) \geq f(O_j \cup \{o\}) - f(O_j)$; and (2) $f(O_i) \leq f(O_j)$. $\square$

Example 5: Assume that each object $o \in O$ is associated with a set of class labels (e.g., Greek restaurant), denoted by $L(o)$. Function $f(X) = |\bigcup_{o \in X} L(o)|$ is a submodular monotone function, where $X \subseteq O$. For example, consider 3 objects o_1, o_2 , and o_3 and their associated labels $L(o_1) = \{a, b, c\}$, $L(o_2) = \{a, d\}$, $L(o_3) = \{c, d\}$. Let $O_1 = \{o_2\}$, and $O_2 = \{o_2, o_3\}$. If we add o_1 into O_1 , we can get an increase in the score of $f(O_1 \cup \{o_1\}) - f(O_1) = 2$. If we add o_1 into O_2 , we can get an increase in the score of $f(O_2 \cup \{o_1\}) - f(O_2) = 1$. Adding o_1 to the subset O_1 can cause a higher increase in the aggregation score. $\square$

Note that the submodular functions have a natural diminishing returns property, i.e. the marginal gain from adding an element to an input set decreases as the size of the input set increases. Thus, the submodular functions are suitable for many applications. Moreover, submodular functions are more general than the aggregate functions like SUM and MAX.

Based on the above definition of submodular monotone function f , we now formally define the *best region search (BRS)* problem:

Definition 2: BRS problem. Given a set of spatial objects O , a submodular monotone function $f : 2^O \rightarrow \mathbb{R}$, and the size $a \times b$ of query rectangle, the **best region search (BRS) problem** is to find a location p from the entire space P such that the aggregate score of the spatial objects in the $a \times b$ rectangular region centered at p , $f(O_{r_p^{a,b}})$, is

maximized:

$$p = \arg \max_{p \in P} f(O_{r_p^{a,b}})$$

□

Without loss of generality, objects on the boundary of rectangles are excluded. For simplicity, we assume that no two objects have the same x coordinate or y coordinate. In real-life, it is possible that several spatial objects share the same x or y coordinate. Our proposed techniques can be easily extended to the case when some objects have the same x coordinate or y coordinate. To handle the extension, we just need to assign a strict order on all spatial objects and always use this order to break ties. For example, consider two spatial objects o_i and o_j . We draw two $a \times b$ rectangles centered at o_i and o_j , denoted by r_i and r_j respectively. Assume that o_i and o_j have the same x -coordinate and we assign an order such that $o_i \succ o_j$. When we scan from left to right, the sweep-line always meets the left edge of r_i before it meets the left edge of r_j .

3.2 An Exact Solution

Intuitively, the *BRS* problem requires us to select a point from infinite points in space P . However, the naive approach of scanning each point in P to find a solution to this problem is prohibitively expensive. Hence, in this section we present an efficient exact solution to the *BRS* problem to address this challenge. To this end, we first introduce the *submodular weighted rectangle intersection (SIRI)* problem and show that we can reduce the *BRS* problem to the *SIRI* problem. Next, we discuss the *SliceBRS* algorithm to solve the *SIRI* problem (and thus *BRS* problem).

3.2.1 The SIRI Problem

We now give the formal definition of the *SIRI* problem. In the next subsection, we shall present an interesting property of the *SIRI* problem, which provides the justification behind our goal to reduce the *BRS* problem to the *SIRI* problem.

Definition 3:SIRI Problem. Consider a set of rectangles of size $a \times b$, R , and a submodular monotone function $h : 2^R \rightarrow \mathbb{R}$ that maps a set of rectangles to a real number. We say a rectangle r is **affected** by a point p if p is inside r . We denote by $A(p)$ the set of rectangles that are *affected* by p . The **weight** of point p is defined by the submodular monotone function h on the set of rectangles $A(p)$, which is denoted by $h(A(p))$. The **SIRI problem** aims at finding a point p with the maximum weight in space P :

$$p = \arg \max_{p \in P} h(A(p)).$$

 Figure 3.1: Reduction from *BRS* to *SIRS*.

□

The *BRS* problem can be reduced to the *SIRS* problem as follows. For each spatial object $o \in O$, we draw an $a \times b$ rectangle $r_o^{a,b}$ centered at o . For a set of rectangles $R_i = \{r_{o_1}^{a,b}, \dots, r_{o_i}^{a,b}\}$, let $h(R_i) = f(\{o_1, \dots, o_i\})$. Therefore, we get an instance of the *SIRS* problem. We illustrate the reduction with an example. Consider the example in Figure 3.1. In Fig 3.1(a), O comprises 4 spatial objects (each represented by a black dot) in space P . The size of the query rectangle is $a \times b$. To reduce the *BRS* problem to the *SIRS* problem, for each spatial object $o_i \in O, i \in [1, 4]$, we draw an $a \times b$ rectangle centered at o_i , denoted by r_i , as depicted in Fig 3.1(b).

To prove the answer to the *SIRS* problem is also an answer to the *BRS*, we first present the following lemma.

Lemma 1: Consider a point p and a spatial object o in space P . Object o is inside the $a \times b$ rectangular region $r_p^{a,b}$ iff p is inside the $a \times b$ rectangle $r_o^{a,b}$ that centers at o . □

Proof: If o is inside the $a \times b$ rectangular region $r_p^{a,b}$, we have, $p.x - \frac{b}{2} < o.x < p.x + \frac{b}{2}$ and $p.y - \frac{a}{2} < o.y < p.y + \frac{a}{2}$. We can easily derive: $o.x - \frac{b}{2} < p.x < o.x + \frac{b}{2}$, $o.y - \frac{a}{2} < p.y < o.y + \frac{a}{2}$. Thus p is inside the rectangular region $r_o^{a,b}$. Similarly, we can prove that if p is inside the $a \times b$ rectangular region $r_o^{a,b}$, then o is inside the rectangular region $r_p^{a,b}$. Putting these together, we conclude that object o is inside $r_p^{a,b}$ iff p is inside $r_o^{a,b}$.

As shown in Figure 3.1, point p is inside the $a \times b$ rectangle centered at o_1 and o_1 is also inside the rectangle centered at p .

Then we can have the following theorem.

Theorem 1: Given an instance of the *BRS* problem and the instance of the *SIRS* problem which is reduced from the *BRS* problem, an answer to the *SIRS* problem is an answer to the *BRS* problem. □

Proof: Let P be the space in the *BRS* problem and the *SIRS* problem and $p \in P$ is a point in space P . According to Lemma 1, for any spatial object $o_i \in O$, we have $o_i \in O_{r_p^{a,b}}$

iff $r_{o_i}^{a,b} \in A(p)$. Thus $f(O_{r_p^{a,b}}) = h(A(p))$ holds. If $p \in P$ can maximize $h(A(p))$ in the *SIRI* problem, then it can also maximize $f(O_{r_p^{a,b}})$ in the *BRS* problem. Consequently, an answer to the *SIRI* problem is also an answer to the *BRS* problem.

Note that the reduction is inspired by the idea of reducing the *MaxRS* problem to the rectangle intersection problem [NB95, CCT12]. Given a set of weighted rectangles, the rectangle intersection problem aims at finding an area such that the sum of the weights of rectangles that intersect with this area is maximized. However, the techniques designed for solving the rectangle intersection problem cannot be used to solve the *SIRI* problem.

3.2.2 Disjoint Regions

Next, we present an interesting property of the *SIRI* problem, which also enables us to justify the reason behind the reduction of the *BRS* problem to the *SIRI* problem and solve the reduced *SIRI* problem instead of solving the *BRS* problem directly.

Definition 4: Disjoint Regions. The edges of the rectangles in R divide the space P into regions. We say a region τ_i is a **disjoint region** iff: (1) τ_i is the intersection of a set of rectangles R_i ; (2) there exists no region τ_j such that (a) τ_j is the intersection of a set of rectangles R_j , (b) $R_i \subseteq R_j$, and (c) $\tau_j \subseteq \tau_i$. We denote the set of rectangles whose intersection is τ_i as $\tau_i.R_i$. $\square$

Example 6: Consider the example in Fig 3.2. There are four rectangles, r_1, r_2, r_3 and r_4 in the space. The four rectangles together divide the space into 12 disjoint regions, each is marked with an ID. Disjoint region τ_9 (filled with grey color) is the intersection of the set of rectangles $\{r_2, r_3\}$. $\square$

Lemma 2: All the points in a disjoint region can affect the same set of rectangles, whose intersection forms the disjoint region. $\square$

Proof: Since each disjoint region is the intersection of a set of rectangles R_i , all points in the disjoint region can affect all rectangles in R_i .

Theorem 2: There are at most $O(n^2)$ disjoint regions, where n is the number of rectangles in R . $\square$

Proof: We draw $2 \cdot n$ horizontal lines passing the horizontal edges of the n rectangles and $2 \cdot n$ vertical lines passing the vertical edges of the n rectangles. The horizontal lines and the vertical lines divide the space into $O(n^2)$ cells. Each disjoint region comprises at least one cell. Thus there are at most $O(n^2)$ disjoint regions.

Observe that for both the *BRS* problem and the *SIRI* problem, we find a point from the infinite points in space P . Lemma 2 and Theorem 2 together enable us to significantly reduce the search space. According to Lemma 2, instead of considering the infinite points

Figure 3.2: Disjoint regions.

 Figure 3.3: $O(n^2)$ maximal regions

in space P , we only need to consider a point for each disjoint region. Theorem 2 tells us there are at most $O(n^2)$ regions to be considered. Consequently, by reducing the *BRS* problem to the *SIRI* problem, we convert our problem from selecting a point from infinite points in space P to selecting a disjoint region from $O(n^2)$ disjoint regions.

3.2.3 Disjoint Region-based Solution

We proceed to present an efficient approach to solve the *SIRI* problem using the notion of disjoint region. To solve the *SIRI* problem, based on Lemma 2 and Theorem 2, a straightforward strategy is outlined as follows: we find all disjoint regions, and for each disjoint region $\mathbf{r}_i$ and the set of rectangles, denoted by $\mathbf{r}_i.R_i$, that are affected by points in region $\mathbf{r}_i$, we compute its weight, denoted by $h(\mathbf{r}_i.R_i)$. After we find the disjoint region with maximum weight, any point in the region can be an answer to the *SIRI* problem. However, there are $O(n^2)$ disjoint regions in the worst case and it is inefficient to examine every disjoint region. The question here is whether we need to examine all the disjoint regions. We next use an example to show that we do not need to examine every disjoint region.

Example 7: Consider the example in Fig 3.2. The points in disjoint region $\mathbf{r}_3$ can affect $\{r_1, r_4\}$ and the points in disjoint region $\mathbf{r}_5$ can affect $\{r_1, r_3, r_4\}$. The rectangles affected by the points in $\mathbf{r}_3$ is a subset of the rectangles affected by the points in $\mathbf{r}_5$. Since h is a submodular monotone function, $h(\{r_1, r_4\}) \leq h(\{r_1, r_3, r_4\})$ always holds. Thus there is no need to examine $\mathbf{r}_3$. $\square$

Inspired by the above example, we propose the notion of *maximal region* and show that we only need to check the maximal regions instead of all disjoint regions.

Definition 5:Maximal Region. Consider a set of rectangles R . The edges of the rectangles divide the space into a set of disjoint regions $\mathfrak{R}$. A disjoint region $\mathbf{r} \in \mathfrak{R}$ is defined as a *maximal region* if:

Figure 3.4: Example of maximal slabs.

Figure 3.5: Find maximal slabs.

- (1) τ is a rectangular region;
- (2) τ 's left edge is part of the left edge of a rectangle in R ;
- (3) τ 's right edge is part of the right edge of a rectangle in R ;
- (4) τ 's top edge is part of the top edge of a rectangle in R ;
- (5) τ 's bottom edge is part of the bottom edge of a rectangle in R .

□

Example 8: In Fig 3.2, τ_5 is a maximal region because (1) it is a rectangular region; (2) its left edge is part of the left edge of rectangle r_3 ; (3) its right edge is part of the right edge of rectangle r_1 ; (4) its top edge is part of the top edge of rectangle r_4 ; and (5) its bottom edge is part of the bottom edge of rectangle r_1 . As another example, τ_8 is also a maximal region. However, τ_3 is not a maximal region because its right edge is not part of the right edge of any rectangle, but is part of the left edge of rectangle r_3 . As another example, τ_1 is not a maximal region because it is not a rectangular region. □

Lemma 3: For any disjoint region τ_i , there exists a maximal region τ_j such that $A(p_{\tau_i}) \subseteq A(p_{\tau_j})$, where p_{τ_i} is a point from τ_i and p_{τ_j} is a point from τ_j . □

Proof: For a maximal region τ , if there exists no horizontal line between the top and bottom edges of the region, then the area between the two horizontal lines passing through the top and bottom edges of the region is a maximal slab. If there exist horizontal lines between the top and bottom edges of the region, then there exists at least one maximal slab between the top and bottom edges of the region.

From Lemma 3, we know that to find a point p with maximum weight $h(A(p))$, we only need to check the points in the maximal regions instead of all disjoint regions. Usually, the number of maximal regions is much smaller than the number of disjoint regions. However, in the worst case, there are still $O(n^2)$ maximal regions.

Lemma 4: In the worst case, there are $O(n^2)$ maximal regions, where n is the number of rectangles. $\square$

3.2.4 Maximal Slab-based Solution

Observe that the solution proposed in the preceding subsection requires us to examine all maximal regions in order to find *only one* region that is the answer to the *SIRI* problem. Thus, a natural question is whether we have to examine *all* maximal regions to find this region? In this subsection, we propose a novel approach to enable us to prune the search space of examining maximal regions.

Our idea is based on the concept of *maximal slab*. Each maximal region intersects at least one maximal slab. We show that there are at most $O(n)$ maximal slabs, where n is the number of rectangles. Furthermore, we show that we can compute an upper bound for each maximal slab and prune the maximal slabs based on the estimated upper bounds. Consequently, the maximal regions that overlap with the pruned maximal slabs are pruned.

Definition 6:Maximal Slab. Consider a set of rectangles R in space P . A **maximal slab** is the area between two horizontal lines in the space where (1) the top horizontal line passes the top edge of a rectangle; (2) the bottom horizontal line passes the bottom edge of a rectangle; and (3) the area between the two horizontal lines does not contain top or bottom edge of any rectangle. $\square$

Example 9: We use the example in Figure 3.5 to illustrate maximal slab. For each rectangle, we extend its top edge and bottom edge to get two horizontal lines, as shown in Figure 3.5(b). The vertical edges are ignored. We use solid line for the horizontal line passing the top edge of a rectangle and dashed for the bottom edge. The grey-filled slab s_1 in Figure 3.5 (b) is the only maximal slab. $\square$

Based on the definition of maximal slab, we can derive the following lemma:

Lemma 5: For any maximal region $\mathbf{r}$, there exists at least one maximal slab s such that $\mathbf{r}$ intersects with s . $\square$

Proof: For a maximal region $\mathbf{r}$, if there exists no horizontal line between the top and bottom edges of the region, then the area between the two horizontal lines passing through the top and bottom edges of the region is a maximal slab. If there exists horizontal lines between the top and bottom edges of the region, then there exists at least one maximal slab between the top and bottom edges of the region.

Lemma 6: There are at most $O(n)$ maximal slabs, where n is the number of rectangles. $\square$

Proof: There are at most n horizontal lines passing the top edge of a rectangle and each of them can be paired with zero or one horizontal line passing the bottom edge of a rectangle. Thus there are at most n maximal slabs.

Lemma 5 guarantees that we will not miss the answer to the *SIRI* problem if we only search inside the maximal slabs. Inspired by this, we only consider the space in all maximal slabs and search for the answer to the *SIRI* problem in each maximal slab. We can compute an upper bound for each maximal slab and prune the maximal slabs using the estimated upper bounds.

Lemma 7: Given a maximal slab s , let R_s denote the set of rectangles that intersect with s . For any point p in s , $h(R_s)$ is an upper bound of $h(A(p))$, denoted as $upper(s)$. $\square$

Proof: Since R_s is the set of rectangles that intersect with s , for any point p in s , we have $A(p) \subseteq R_s$. As the function h is submodular and monotone, $h(R_s) \geq h(A(p))$ holds.

Example 10: Consider the maximal slab s_2 in Fig 3.4. We compute $upper(s_2) = h(\{r_1, r_2, r_4\})$. For any point p in s_2 , we have $h(A(p)) \leq upper(s_2)$. $\square$

Approach We now present how to utilize maximal slabs to efficiently find an exact answer to the *SIRI* problem. Our approach broadly comprises two steps. First, we find the maximal slabs by using a sweep line to scan the space bottom-up to find the set S of maximal slabs together with their upper bounds. Second, we find the exact answer by checking the maximal slabs s in descending order of their upper bounds and searching for the point with maximum weight in each processed maximal slab. Once the upper bound of any remaining maximal slab in S is smaller than the best known result, we stop checking the remaining maximal slabs. We elaborate on these steps in turn.

Finding maximal slabs The main idea is to use a sweep line to scan bottom-up. While scanning, we maintain the edges that the sweep line has encountered. According to the definition of maximal slab, if the sweep line encounters a bottom edge and a top edge consecutively, then a maximal slab is found. The outline of the procedure is reported in Function `ScanSlab`. The function takes as input the set of rectangles R and returns all maximal slabs S in the space. It uses a horizontal line, denoted by l , to sweep bottom-up, and uses a queue Rec to store the rectangles that the sweep line has met. It uses $flag$ to denote whether the last edge that l met is a top edge of a rectangle or a bottom edge and $flag$ is initialized by **bottom**. When the sweep line l encounters a bottom edge of a rectangle r , it pushes r into Rec (line 4) and mark $flag$ as **bottom** (line 5). When the sweep line l encounters a top edge of a rectangle r , if $flag$ is **bottom**, it finds a new maximal slab with upper bound $h(Rec)$ and add it to S (lines 8–9). Next, it removes r from Rec and mark $flag$ as **top** (lines 10–11). The algorithm terminates when the sweep line reaches the top of the space.

Function ScanSlab(R)

Output: A list of maximal slabs S .

```

1   $Rec \leftarrow \emptyset, S \leftarrow \emptyset, flag \leftarrow \text{bottom};$ 
2  while Sweeping the horizontal line from bottom to top do
3      if  $l$  meets the bottom edge of a rectangle  $r$  then
4           $Rec \leftarrow Rec \cup \{r\};$ 
5           $flag \leftarrow \text{bottom};$ 
6      else
7           $/*$  The sweep line meets the top edge of a rectangle  $*/$ 
8          if  $flag = \text{bottom}$  then
9               $s \leftarrow \text{new maximal slab}, s.UB = h(Rec);$ 
10              $S \leftarrow S \cup \{s\};$ 
11              $Rec \leftarrow Rec \setminus \{r\};$ 
12              $flag \leftarrow \text{top};$ 
13 return  $S;$ 
    
```

Example 11: Suppose we invoke the ScanSlab procedure on the example shown in Figure 3.4(b). A horizontal line sweeps from bottom to top. It first comes across the bottom edge of r_3 and r_3 is added into the tail of Rec . The $flag$ is marked as **bottom**. Then the bottom edges of r_2 and r_1 are processed consequently and $Rec = [r_3, r_2, r_1]$, $flag = \text{bottom}$. After that, the sweep line comes across the top edge of r_3 . Since $flag = \text{bottom}$, the algorithm finds a maximal slab s_1 . We assume $h(\{r_1, r_2, r_3\}) = 5$. The algorithm then pushes s_1 into S . Then r_3 is removed from Rec and $Rec = [r_2, r_1]$ and $flag$ is marked as **top**. The sweep line keeps moving and it meets the bottom edge of r_4 . The algorithm adds r_4 into Rec and mark $flag$ as **bottom** again. Then it meets the top edge of r_2 and $Rec = [r_2, r_1, r_4]$, $flag = \text{bottom}$. Another maximal slab s_2 is found. We assume that $h(\{r_2, r_1, r_4\}) = 4$ and s_2 are pushed into S . The algorithm terminates when the sweep line meets the top edge of r_4 . $\square$

Finding the answer In the second step, we search for the answer iteratively. In each iteration, we pop out the maximal slab s with maximum upper bound. If the upper bound of s is smaller than the current result, we can conclude that the current result is the final result. Otherwise, we process the maximal slab as follows: We use a vertical sweep line to scan s from left to right. During the scan, we maintain the edges that the sweep line has encountered. If the sweep line encounters a left edge and a right edge consecutively, then the area between the two edges may belong to a maximal region and we check whether it has a better weight than the current result.

The formal algorithm is outlined in Function **SearchMR**. We use p_c to store the current answer to the problem and it is initialized as **null**(line 1). It keeps searching while there exists a maximal slab $s \in S$ such that $upper(s) > h(A(p_c))$ (lines 2–14). Specifically, it

Function SearchMR(R, S)

Output: A point p

```

1   $p_c \leftarrow null$ ;
2   $s \leftarrow \arg \max_{s \in S} s.UB$ ;
3  while  $s.UB > h(A(p_c))$  do
4       $flag \leftarrow left, Rec \leftarrow \emptyset$ ;
5      while Sweeping the vertical line  $l$  from left to right in  $s$  do
6          if  $l$  meets the left edge of a rectangle  $r$  then
7               $flag \leftarrow left, Rec \leftarrow Rec \cup \{r\}$ ;
8          else
9              if  $flag = left$  then
10                  $p \leftarrow \text{midpoint of } l \cap s$ ;
11                 if  $h(A(p)) > h(A(p_c))$  then
12                      $p_c \leftarrow p$ ;
13                  $Rec \leftarrow Rec \setminus \{r\}, flag \leftarrow right$ ;
14   $s \leftarrow \arg \max_{s \in S} s.UB$ ;
15 return  $p_c$ ;
    
```

finds the maximal slab s with maximum upper bound in S (line 2) and uses a vertical line l to sweep from left to right (lines 5–14). The variable $flag$ is used to denote whether the last edge that the sweep line l encountered is a left edge or a right edge and it is initialized as *left*. The variable Rec denotes the rectangles that intersect with the sweep line l (line 4). While the vertical sweep line l meets the left edge of rectangle r , the algorithm sets $flag$ as *left* and adds r into Rec (lines 6–7). Otherwise, if the last edge that l met is a left edge (line 9), it takes the midpoint of segment $l \cap s$ (line 10). It then compares its weight and the current result and update p_c if it has a larger weight (lines 11–12). Then r is removed from Rec and the algorithm sets $flag$ as *right* (line 13). When the upper bounds of the maximal slabs in the remaining S are smaller than the current result, it returns p_c as the answer to the *SIRI* problem.

3.2.5 Search Space Reduction by Slicing

We next present a strategy to further reduce the search space. Our main idea can be explained as follows: We divide the space into slices with equal width along the x -axis and process each slice separately by following the divide-and-conquer strategy. We estimate an upper bound for each slice and prune the slices whose upper bound is smaller than the current result.

To realize our idea, we need to answer the following question: *How to set the width of the slices?* Observe that if the width of each slice is too small, then each rectangle will intersect with many slices. In other words, we need to consider a rectangle in many

slices, which yields redundant computation. If the width of each slice is too large, then there will be fewer slices, and thus it is likely that only few slices can be pruned by our strategy. Therefore, it is important to set an appropriate width of the slices. However, if we set the width as a constant independent of the query, then given a rectangle r , the number of slices that intersect with r is dependent on the query rectangle, and is not bounded. Therefore, we may consider a rectangle unbounded number of times (for each slice that it intersects), and this incurs redundant computation. To this end, we propose to set the width of slices to be query dependent. Specifically, we set the width of each slice as θb , where b is the width of the query rectangle and θ is a positive real constant, which can be set empirically (as we shall see in Section 3.4).

Example 12: Consider the example in Fig 3.6. The width of each slice is θb and $\theta > 1$. r_3 intersects with slice 1 and slice 2. $\{r_1, r_3\}$ is the set of rectangles that intersect with slice 1. $\{r_2, r_3, r_4\}$ is the set of rectangles that intersect with slice 2. $\square$

We can derive the following lemma:

Lemma 8: Let the width of each slice be θb . Each rectangle r intersects with at most $\lceil \frac{1}{\theta} \rceil + 1$ slices. $\square$

Proof: We project each rectangle to the x -axis and get a line segment with a length of b . Similarly, we project each slice to the x -axis and get a line segment with a length of θb . A rectangle intersects with a slice iff their projections on x -axis intersect. The projection of a rectangle can intersect at most $\lceil \frac{1}{\theta} \rceil + 1$ projections of slices. Thus, each rectangle r intersects with at most $\lceil \frac{1}{\theta} \rceil + 1$ slices.

Lemma 8 guarantees that each rectangle is processed a constant number of times if we process each slice by invoking the **ScanSlab** procedure. Thus, we further have:

Lemma 9: There are at most $(\lceil \frac{1}{\theta} \rceil + 1)n$ maximal slabs in all slices, where n is the number of rectangles. $\square$

Proof: Since each rectangle intersects at most $\lceil \frac{1}{\theta} \rceil + 1$ slices, the top edge of a rectangle can contribute to at most $\lceil \frac{1}{\theta} \rceil + 1$ maximal slabs. Thus there are at most $(\lceil \frac{1}{\theta} \rceil + 1)n$ maximal slabs in all slices, where n is the number of rectangles.

According to Lemma 9, it still takes $O(n)$ time to find maximal slabs in all slices, where n is the number of rectangles. Note that we can find the maximal slabs S_i in slice i by invoking the **ScanSlab** procedure in slice i .

3.2.6 The SliceBRS Algorithm

We now have all the machinery in place to discuss the *SliceBRS* algorithm to address the *BRS* problem. The key idea of the algorithm is as follows. We first transform the *BRS*

Algorithm 1: SliceBRS

Input: A set of objects O , query size $a \times b$, aggregate function f

Output: A point p

```

1   $p_c \leftarrow \text{null}$ ;
2   $R \leftarrow$  the set of  $a \times b$  rectangles centered at each  $o \in O$ ;
3  Divide the space into slices with width  $\theta b$ ;
4  for each slice  $i$  do
5       $R_i \leftarrow$  the set of rectangles that intersect with slice  $i$ ;
6       $S_i \leftarrow \text{ScanSlab}(R_i)$ ;
7  while there exists a slice  $i$  s.t.  $\max_{s \in S_i} s.UB > h(A(p_c))$  do
8       $p \leftarrow \text{SearchMR}(R_i, S_i)$ ;
9      if  $h(A(p)) > h(A(p_c))$  then
10          $p_c \leftarrow p$ ;
11 return  $p_c$ ;
    
```

Figure 3.6: Equal-width slices.

Figure 3.7: c -cover

problem into the *SIRI* problem and divide the space vertically into several slices with a fixed width. Next, in each slice, the algorithm performs a fast check and finds the list of maximal slabs together with their upper bounds. It uses the maximum upper bound of the slabs in a slice as the upper bound of the slice. Then, it greedily processes slices. The slice with a larger upper bound is processed first. In each slice, the algorithm searches each maximal slab greedily. A maximal slab with a larger upper bound is searched first. The algorithm terminates when the weight of the known best point is larger than the upper bound of the unprocessed slices, or all the slices have been processed.

The *SliceBRS* algorithm is presented in Algorithm 1. It takes as input a set O of spatial objects, an aggregate score function f , and the size $a \times b$ of a query rectangle. It returns the point p with maximum $h(A(p))$. It uses p_c to maintain the point with maximum weight that has currently been found and it is initialized by **null** (line 1). The algorithm first transforms the *BRS* problem to the *SIRI* problem by drawing an $a \times b$

rectangle centered at o for each spatial object $o \in O$ (line 2). Then it vertically divides the space into slices with width θb (line 3). After that, for each slice i , it estimates the upper bound and finds maximal slabs in the slice (lines 4–6). Based on the upper bounds, the algorithm processes the slices that are most likely to contain the best point and update p_c once a better point is found (lines 7–10). The algorithm terminates when the upper bound of the remaining slices is worse than the current result.

Complexity The *SliceBRS* algorithm comprises three phases: (1) Cutting the space into slices, (2) Scanning maximal slabs, and (3) Searching in maximal slabs. We analyze the time complexity of the three phases in turn. Let n be the number of spatial objects in O . In phase 1, it takes $O(n)$ time to cut the space into slices. In phase 2, according to Lemma 9, there are at most $(\lceil \frac{1}{\theta} \rceil + 1)n$ maximal slabs in all slices, where θ is a predefined constant. Thus, it takes Function **ScanSlab** $O(n)$ time to process all slices. In phase 3, it takes Function **SearchMR** $O(n)$ time to search one maximal slab. Since we can safely ignore the maximal slabs whose upper bounds are lower than the current result, we assume only n_s maximal slabs are actually searched by Function **SearchMR**. Thus the complexity of phase 3 is $O(n \cdot n_s)$. Putting these together, we can conclude that in the worst case, it takes $O(n + n + n \cdot n_s) = O(n \cdot n_s)$ time to find the best point.

Although n_s can be n in the worst case, n_s is usually a very small value. To have a better idea, we consider the complexity of *SliceBRS* for the following case.

Lemma 10: Consider a set O consisting of n spatial objects in a $W \times W$ space P . We assume that the objects in O are uniformly distributed in space P . The complexity of *SliceBRS* for this case is $O(n)$. $\square$

Proof: Since the objects in O are uniformly distributed in space P , there are $\sqrt{n}$ spatial objects in a row and a column. The distance d_h between two horizontal adjacent spatial objects, as well as the distance d_v between two vertical adjacent objects, are given by $d_v = d_h = \frac{W}{\sqrt{n}}$. Let the size of the query rectangle be $a \times b$. We reduce the *BRS* problem to the *SIRI* problem by drawing an $a \times b$ rectangle centered at o for each object $o \in O$. We still analyze the complexity of the 3 phases of *SliceBRS* in turn: It takes $O(n)$ time to cut the space into slices in phase 1, and $O(n)$ time to find all maximal slabs in phase 2. In phase 3, we first consider the complexity of Function **SearchMR** searching one maximal slab. Recall that there are $O(\sqrt{n})$ spatial objects in a row in the space. We draw a rectangle centered at o for each spatial object o and thus there are $O(\sqrt{n})$ left edges and $O(\sqrt{n})$ right edges. Since the space is equally divided into $\frac{W}{\theta b}$ slices, there are $O(\sqrt{n} \div \frac{W}{\theta b}) = O(\frac{\theta b \cdot \sqrt{n}}{W})$ left edges and right edges in one slice. Therefore, it takes $O(\frac{\theta b \cdot \sqrt{n}}{W})$ time for Function **SearchMR** to process one maximal slab. There are $O(\sqrt{n})$ maximal slabs in one slice and $\frac{W}{\theta b}$ slices in the space. Thus, it takes $O(\frac{\theta b \cdot \sqrt{n}}{W} \cdot \sqrt{n} \cdot \frac{W}{\theta b}) = O(n)$ time to find the best point. Putting these together, we can conclude that the complexity of *SliceBRS* for this case is $O(n)$.

In practice, according to our experiments, usually only a few of maximal slabs are actually searched and n_s is a very small value (from 30 to 3000).

3.3 Approximate Solution

Although the *SliceBRS* algorithm can efficiently find an exact answer to the *BRS* problem, we observed that its runtime performance degrades when the number of spatial objects grows or the spatial objects in the space get denser (detailed in Section 3.4). Hence in this section, we present a constant-bounded approximate algorithm called *CoverBRS* for answering the *BRS* problem efficiently especially for large datasets.

3.3.1 Overview

The key idea behind our approximate algorithm for solving the *BRS* problem is as follows: we select a set of points, denoted by T , from the space P so that each point in T can represent some spatial objects in O . The points in T together preserve some properties of O such that the rectangle region found on the set of objects T can be an approximation of the result on O with performance guarantees. This idea has two potential benefits: (1) T contains fewer spatial objects than O ; (2) The spatial objects in T are sparser than those in O . These benefits enable us to answer the *BRS* problem more efficiently.

How to select and utilize the set T such that we can achieve a bounded approximate answer? To address this challenge, we propose a novel concept called *c-cover* of O , where c is a parameter to control the approximation ratio of our proposed solution. The concept lays the foundation of our proposed algorithm called *CoverBRS*. Specifically, we first select a *c-cover* T of O . Next, we generate a new instance of *BRS* problem for the *c-cover* T by defining a new aggregate score function f_T and a new size $(1-c)a \times (1-c)b$ of the query rectangle. Lastly, we invoke the *SliceBRS* algorithm to solve the new instance and prove that the answer to the new instance of *BRS* problem is a constant-bounded approximate answer to the original one. In the subsequent subsections, we elaborate on these issues.

3.3.2 *c-cover*

We first introduce the concept of *c-cover*. Then we show some properties of *c-cover* which we shall use to achieve a constant-bounded approximate answer to the *BRS* problem.

Definition 7:*c-cover*. Consider a set of spatial objects O in space P and a query rectangle of size $a \times b$. Let T be a set of spatial points in P . We say T is a ***c-cover*** of O iff for any object $o_i \in O$, there exists one point $t \in T$ such that o_i is inside the $ca \times cb$ rectangular region centered at t . $\square$

Hereafter, we refer to the points in T as spatial objects.

Example 13: Consider the example shown in Fig 3.7. The set of black and white nodes is a set of spatial objects, denoted by O . The set of black nodes $\{o_1, o_2, o_3\}$ is a c -cover of O . $\square$

Based on the definition of c -cover, we can derive the following lemma, which is the foundation to get a constant-bounded approximate answer to the *BRS* problem. Based on this lemma, we shall establish the approximation bound of our proposed algorithm in Section 3.3.6.

Lemma 11: Consider a set of spatial objects O , a query rectangle of size $a \times b$. Let T be a c -cover of O , where $c \in (0, 1)$. For each spatial object $t \in T$, let $O_{r_t^{ca,cb}}$ be the set of spatial objects inside the $ca \times cb$ rectangular region centered at t . Given a point p in space P , if a $(1 - c)a \times (1 - c)b$ rectangular region centered at p can cover t , then the $a \times b$ rectangular region centered at p can cover all objects in $O_{r_t^{ca,cb}}$. $\square$

Proof: Let o_i be an object in $O_{r_t^{ca,cb}}$. We have $|o_i.x - t.x| < \frac{cb}{2}$, $|o_i.y - t.y| < \frac{ca}{2}$. Since t is inside a $(1 - c)a \times (1 - c)b$ region centered at p , we have $|t.x - p.x| < (1 - c)b/2$, $|t.y - p.y| < (1 - c)a/2$. Obviously, $|o_i.x - p.x| < \frac{cb}{2}$, $|o_i.y - p.y| < \frac{ca}{2}$. Therefore, o_i is inside the $a \times b$ rectangular region centered at p .

3.3.3 Selection of c -cover

We next study how to select a c -cover of O . Ideally, we would like to select a minimum c -cover, on which we define a new instance of the *BRS* problem. This is because it would be more efficient for answering the new instance defined on a smaller set of spatial objects.

Theorem 3: Finding a minimum c -cover is NP-hard. $\square$

Proof: Consider the geometric version of set cover problem. Given a finite set of points and a set of rectangles with predefined size in a 2 dimensional space, the problem is to find the minimum number of rectangles that cover all points. This problem is NP-hard even when the rectangles are identical squares with their sides parallel to the axes[FPT81]. This problem is a special case of finding a minimum c -cover when the query rectangle is a square. Thus, finding a minimum c -cover is NP-hard.

Since finding the minimum c -cover is a special case of set cover problem, at first glance it may seem that we can approximate the minimum c -cover by using a greedy algorithm with an approximate ratio of $1 - 1/e$, where e is the Euler's number. Specifically, in each iteration, we select the spatial object t that can maximize the number of uncovered objects inside the $ca \times cb$ rectangular region centered at t , referred to as neighborhood of t .

Although the greedy algorithm works, unfortunately its complexity is high. It involves too many range queries in the computation. A range query in 2D space can be answered in $O(\log n + k)$ time[Cha86], where n is the number of spatial objects and k is the number of spatial objects in the query range. In the worst case, there will be $O(n)$ iterations in the greedy algorithm and we have to perform $O(n)$ range queries in each iteration. The greedy algorithm takes $O(n^2 \log n)$ time to select a c -cover in the worst case. Thus, we need to devise a more efficient solution to select a c -cover.

Quadtree-based c -cover selection We propose a quadtree-based heuristic algorithm in order to select a c -cover efficiently. Before we present the algorithm, we first explain how we use the quadtree to index the spatial objects and give a lemma, which enables us to select a relatively small c -cover efficiently.

We use a quadtree to index all spatial objects in the space. The quadtree recursively partitions the space into four equal-sized rectangular regions¹ until each leaf node contains one spatial object. Each node v in the quadtree corresponds to a rectangular region together with the spatial objects in the region. For each node v in the quadtree, we maintain a point, denoted by $v.t$. Specifically, if v is an internal node, let $v.t$ be the center point of the corresponding region. If v is a leaf node, let $v.t$ be the spatial object in the corresponding region.

Example 14: Consider the example in Figure 3.8. We consider five spatial objects $o_1, \dots, o_5$ in the space. We construct a quadtree to index these five spatial objects as shown in the figure. Node v_1 corresponds to the entire space and $v_1.t$ is the center of the entire space. Node v_3 corresponds to the top-right rectangular region. It is a leaf node and it contains spatial object o_1 . Thus $v_3.t$ is spatial object o_1 . $\square$

We proceed to derive the following lemma:

Lemma 12: We denote by $v.l$ the level of a node v in the quadtree. Let V_l^1 be the set of maintained points of internal nodes at level l in the quadtree, *i.e.* $V_l^1 = \{v.t | v \text{ is an internal node} \wedge v.l = l\}$. Let V_l^2 be the set of maintained points of leaf nodes at any level not lower than l , *i.e.* $V_l^2 = \{v.t | v \text{ is a leaf node} \wedge v.l \geq l\}$. If the corresponding region of a node at level l can be fully covered by a $ca \times cb$ rectangle, then V_l^1 and V_l^2 together form a c -cover. $\square$

Proof: We consider the following two cases: (1) If a spatial object o is in a leaf node whose level is not lower than l , then $o \in V_l^2$; (2) If a spatial object o is in a leaf node v whose level is lower than l , then we can traverse along the path from v to the root until we find an internal node u at level l . Spatial object o is in the corresponding region of

¹We use a minimum bounding rectangle (MBR) to enclose all spatial objects, thus we refer to the partitions as “rectangular regions” instead of “quadrants”.

 Figure 3.8: Selection of c -cover T .

u . Since the corresponding region of u can be fully covered by a query rectangle, object o is inside a $ca \times cb$ rectangular region centered at $u.t$.

Therefore, for any spatial object $o \in O$, there exists a $t \in V_l^1 \cup V_l^2$ such that o is inside the $ca \times cb$ rectangular region centered at t . We conclude that $V_l^1 \cup V_l^2$ is a c -cover of O .

Example 15: Consider the example in Figure 3.8. Let the size of the dashed-line rectangle be $ca \times cb$. The corresponding region of nodes at level 2 can be fully covered by the dashed-line rectangle. In this case, $\{o_1, o_3, o_5\}$ together with the center of the region in v_6 form a c -cover. $\square$

Lemma 12 tells us that we can select a c -cover by retrieving V_l^1 and V_l^2 once we identify a level l such that the corresponding region of a node at level l can be fully covered by a $ca \times cb$ rectangle. However, there may exist several possible values for level l . Which value should we select? As we have discussed at the beginning of this subsection, we would like to select a c -cover with smaller number of spatial objects. The center of a higher level region can cover more spatial objects than any center of its descendant regions at lower levels. Therefore, to get a smaller c -cover, we use the highest level that can satisfy the condition in Lemma 12. Since the quadtree recursively partitions the space into four equal-sized rectangular regions, we can directly compute the level by:

$$l = \max(\lceil \log_2 \frac{Height}{ca} \rceil, \lceil \log_2 \frac{Width}{cb} \rceil). \quad (3.1)$$

where $Height$ and $Width$ are the height and the width of the entire space, respectively.

Now we are ready to present our quadtree-based heuristic algorithm. The algorithm comprises three steps: (1) identify the highest level l ; (2) collect the centers of regions in all internal nodes at level l ; and (3) collect the spatial objects in all leaf nodes at any level not lower than l . The algorithm is outlined in *Function Select*. It takes as input a set of

Function Select(O, c)

Output: A set of objects T

```

1  $QT \leftarrow$  the indexing quadtree;
2  $T \leftarrow \emptyset$ ;
3  $l \leftarrow \max(\lceil \log_2 \frac{Height}{ca} \rceil, \lceil \log_2 \frac{Width}{cb} \rceil)$ ;
4 foreach internal quadtree node  $v$  s.t.  $v.l = l$  do
5   |  $T \leftarrow T \cup \{v.t\}$ ;
6 foreach leaf quadtree node  $v$  s.t.  $v.l \geq l$  do
7   |  $T \leftarrow T \cup \{v.t\}$ ;
8 return  $T$ ;
```

Algorithm 2: CoverBRS

Input: A set of objects O , query rectangle size $a \times b$, parameter c

Output: A point p

```

1  $T \leftarrow$  Select( $O, c$ );
2  $f_T \leftarrow$  New aggregate function;
3  $p \leftarrow$  SliceBRS( $T, (1 - c)a, (1 - c)b, f_T$ );
4 return  $p$ ;
```

spatial objects O and a parameter c . We use a quadtree QT to index all spatial objects in the space (line 1). The quadtree is independent of the size of the query rectangle and is constructed in advance. Each node in the quadtree is associated with a rectangular region and each leaf node in the quadtree contains only one spatial object. Given a query rectangle of $a \times b$, in order to select a c -cover of the spatial objects O , we compute the highest level l in the quadtree using the Equation 3.1 (line 3). Then for each internal node at level l in the quadtree, we add the center of the associated rectangular region into T (lines 4–5). For each leaf node whose level is not lower than l in the quadtree, we add the object in the node into T (lines 6–7). Consequently, we find a c -cover of O .

Complexity We can compute the level l in $O(1)$ time. We retrieve the center of internal nodes at level l and all leaf nodes whose level is not lower than l in the quadtree. We use a list to maintain all leaf nodes sorted by their level. Thus, it takes $O(n)$ time to select a c -cover T , where n is the number of spatial objects in O . The complexity of our proposed quadtree-based algorithm is much better than the complexity of the aforementioned greedy algorithm.

3.3.4 Generate a New Instance of BRS

We next present how to generate a new instance of the *BRS* problem based on the generated c -cover of O , which is denoted by T . Specifically, we need to (1) define a new

aggregate score function over T , and (2) define a new size of query rectangle. We first introduce how to define the aggregate function over the c -cover T .

Aggregate score function over c -cover To define a new aggregate score function, we need to represent all the objects in O by objects in T . We utilize the quadtree in *Function Select* to decide which objects in O can be represented by an object in T . Specifically, as we have presented, an object t_i in T is selected from a node v in the quadtree. We let t_i represent all spatial objects inside the region in v . Since the objects in T belongs to different nodes in the quadtree, an object o in O will be represented by exactly only one object in T .

Let $D(t_i)$ denote the set of objects represented by t_i for $t_i \in T$. Then we can define the new aggregate score function f_T for the objects in T in the new instance as follows:

Definition 8: New aggregate score function. We define the new aggregate score function f_T as: $f_T : 2^T \rightarrow \mathbb{R}$, which maps a subset of T to a real number such that for any $T_i \subseteq T$ and $T_i = \{t_1, \dots, t_j\}$

$$f_T(T_i) = f(D(t_1) \cup \dots \cup D(t_j))$$

□

One can easily verify that f_T is still a submodular monotone function.

New size of the query rectangle We next present how to define the size of new query rectangle and the intuition behind it. According to Lemma 11, we define the size of query rectangle in the new instance as $(1 - c)a \times (1 - c)b$. Note that if a $(1 - c)a \times (1 - c)b$ rectangular region centered at a point p can cover an object $t_i \in T$ in the new instance, then the $a \times b$ rectangular region centered at p can cover all objects $D(t_i)$ represented by t_i in the original instance. In this way, the answer to the new instance is a good approximation of the answer to the original *BRS* problem. We shall prove later that the answer to the new instance is a constant-bounded approximate answer to the original one. Observe that once we have generated a new instance of the *BRS* problem with the newly defined aggregate score function and the size of the query rectangle, we can invoke the *SliceBRS* algorithm to solve the new instance.

3.3.5 The CoverBRS Algorithm

We are now ready to present the *CoverBRS* algorithm which is outlined in Algorithm 2. It takes as input a set of objects O , the size $a \times b$ of query rectangle, and a parameter c for selecting the c -cover T . It first selects a set T of spatial points from space P by invoking the **Select** procedure with the parameter c (line 1). Then it defines a new aggregate score function, denoted by f_T , based on Definition 8 for the c -cover T (line 2). Lastly, it

invokes the *SliceBRS* algorithm to answer the new instance of the *BRS* problem on the subset T with aggregate function f_T and query rectangle of size $(1 - c)a \times (1 - c)b$.

Complexity It takes $O(n)$ for our quadtree-based heuristic algorithm to select the subset T , where n is the number of spatial objects in O . It takes $O(n^t \times n_s^t)$ time to invoke the *SliceBRS* algorithm to solve the new instance, where n^t is the number of spatial objects in T and n_s^t is the number of maximal slabs that are actually searched. Hence, the overall time complexity of the *CoverBRS* algorithm is $O(n + n^t \times n_s^t)$. Recall that the complexity of *SliceBRS* is $O(n \times n_s)$. Since the c -cover is a subset of all spatial objects and only a few maximal slabs are searched, *CoverBRS* has a better efficiency than *SliceBRS*. In our experiment, we can observe that *CoverBRS* is always more efficient than *SliceBRS*, especially when there are tens to hundreds of millions of spatial objects in the space.

3.3.6 Approximation Ratio

Finally, we present the approximation ratio of the answer returned by the *CoverBRS* algorithm for different values of c and also prove that the approximation ratio is tight.

Lemma 13: Let p be any point in the space, $O_{r_p^{a,b}}$ be the set of objects in O that are covered by $r_p^{a,b}$, and $T_{r_p^{(1-c)a, (1-c)b}}$ be the set of objects in T that are covered by $r_p^{(1-c)a, (1-c)b}$. The following holds:

$$f_T(T_{r_p^{(1-c)a, (1-c)b}}) \leq f(O_{r_p^{a,b}})$$

□

Proof: We have

$$\begin{aligned} f_T(T_{r_p^{(1-c)a, (1-c)b}}) &= f_T(\{t_i, \dots, t_j\}) \\ &= f(D(t_i) \cup \dots \cup D(t_j)) \leq f(O_{r_p^{a,b}}) \end{aligned}$$

Theorem 4: When $c = 1/3$, Algorithm *CoverBRS* returns a $(1/4)$ -approximate answer to the optimal solution. □

Proof: Let p_{opt} be the optimal center point for the $a \times b$ rectangular region with maximum aggregation. Let p be the point returned by the *CoverBRS* algorithm, which is the center point for the $\frac{2}{3}a \times \frac{2}{3}b$ rectangular region with maximum aggregation with respect to the set T . We can find 4 $\frac{2}{3}a \times \frac{2}{3}b$ rectangles $r_1, \dots, r_4$ such that for any object o covered by $r_{p_{opt}}^{a,b}$, its representer is covered by $r_1 \cup \dots \cup r_4$. Consider the Example in Figure 3.9(a), objects $o_1, \dots, o_4$ represent $o_5, \dots, o_8$ respectively and $o_1, \dots, o_4$ are covered by $r_1 \cup \dots \cup r_4$. Thus, we can achieve the following derivation:

$$\begin{aligned} f(O_{r_{p_{opt}}^{a,b}}) &\leq f_T(T_{r_1} \cup \dots \cup T_{r_4}) \\ &\leq f_T(T_{r_1}) + \dots + f_T(T_{r_4}) \leq 4f_T(T_{r_p^{\frac{2}{3}a, \frac{2}{3}b}}), \end{aligned}$$

Figure 3.9: Approximate Ratio.

where T_{r_i} is the set of objects in T that are covered by rectangle r_i for $i \in [1, 4]$.

From Lemma 13, we have

$$f(O_{r_{p_{opt}}^{a,b}}) \leq 4f(O_{r_p^{a,b}})$$

Theorem 5: The $1/4$ approximation ratio is tight for the *CoverBRS* algorithm when $c = 1/3$. $\square$

Proof: We prove this with a worst case example. Consider an instance in Figure 3.9(a) where the solid-line rectangle is a $a \times b$ rectangle and the dashed-line rectangles are $\frac{2}{3}a \times \frac{2}{3}b$ rectangles. The c -cover is $T = \{o_1, o_2, o_3, o_4\}$ and they represent o_5, o_6, o_7 and o_8 , respectively. We assume $f(\{o_1\}) = \dots = f(\{o_4\}) = 1$, $f(\{o_5\}) = \dots = f(\{o_8\}) = 1 - \beta$, where β is a small positive real number, $f(\{o_1, o_5\}) = \dots = f(\{o_4, o_8\}) = 1$, and $f(\{o_4, \dots, o_8\}) = 4 - 4\beta$. In this case, the *CoverBRS* algorithm may choose any of $o_1, \dots, o_4$ as the approximate solution while the optimal solution should be the center of the solid-line rectangle. Since β can be any small value, our answer is $(1/4)$ -approximate.

Theorem 6: When $c = 1/2$, Algorithm 2 returns a $(1/9)$ -approximate answer to the optimal solution. $\square$

Proof: Similar to the proof of Theorem 4, Let p_{opt} be the optimal center point for the $a \times b$ rectangle with maximum aggregation. Let p be the point returned by *CoverBRS* algorithm, which is the center point for the $\frac{1}{2}a \times \frac{1}{2}b$ rectangle with maximum aggregation with respect to the set T of spatial objects. Different from the proof of Theorem 4, we need $9 \frac{1}{2}a \times \frac{1}{2}b$ rectangles $r_1, \dots, r_9$ to guarantee that for any object $o \in O$ covered by

$r_{p_{opt}}^{a,b}$, its representer is covered by $r_1 \cup \dots \cup r_9$, as illustrated in Figure 3.9(b). Thus, we can achieve the following derivation:

$$\begin{aligned} f(O_{r_{p_{opt}}^{a,b}}) &\leq f_T(T_{r_1} \cup \dots T_{r_9}) \\ &\leq f_T(T_{r_1}) + \dots + f_T(T_{r_9}) \leq 9f_T(T_{r_p^{\frac{1}{2}a, \frac{1}{2}b}}) \end{aligned}$$

From Lemma 13, we have

$$f(O_{r_{p_{opt}}^{a,b}}) \leq 9f(O_{r_p^{a,b}}).$$

Theorem 7: The $1/9$ approximation ratio is tight for the *CoverBRS* algorithm when $c = 1/2$. $\square$

Proof: We prove this by giving a worst case example. Consider an instance in Figure 3.9(b) where the solid-line rectangle is a $a \times b$ rectangle and the dashed-line rectangles are $\frac{1}{2}a \times \frac{1}{2}b$ rectangles. The c -cover is $T = \{o_1, \dots, o_9\}$ and they represent $o_{10}, \dots, o_{18}$ respectively. We assume $f(\{o_1\}) = \dots = f(\{o_8\}) = 1$ and $f(\{o_9\}) = \dots = f(\{o_{17}\}) = 1 - \beta$ and $f(\{o_{18}\}) = \beta$ where β is a small positive real number. We assume $f(\{o_1, o_{10}\}) = \dots = f(\{o_8, o_{17}\}) = 1$ and $f(\{o_9, o_{18}\}) = 1 - \beta$. We also assume that $f(\{o_{10}, \dots, o_{18}\}) = 9 - 9\beta$. In this case, since the *CoverBRS* algorithm only consider the objects in the c -cover which locate at $o_1, \dots, o_8$, we may end up choosing any one of $o_1, \dots, o_8$ as the approximate solution. However, the optimal solution should be the center of the solid-line rectangle and its aggregation is $9 - 9\beta$. Since β can be any small positive value, our answer is $(1/9)$ -approximate.

3.4 Experimental Study

In this section, we first present the setup of our experiments. Then, we study the performance of our algorithms w.r.t quality of results, efficiency, and scalability using real-world datasets and applications. All algorithms are implemented in C++ and compiled by VS 2013. All experiments are run on a Windows PC with Intel Xeon 3.70GHz CPU and 64GB memory.

3.4.1 Experimental Setup

Real-world applications Recall from Chapter 1, the *BRS* problem can be exploited to address the problem of (1) finding the most influential region and (2) finding the most diversified region. Hence, we evaluate the performance of our algorithms in the context of these two applications. Here, we briefly describe the setup for these two applications.

Application 1: Most Influential Region. Recall that the goal of this application is to find the most influential region in the context of the influence maximization problem [KKT03a]. Specifically, given the size of a region, we can apply our proposed algorithms to find the most influential region satisfying the size constraint such that the expected number of influenced users is maximized. In this application, we adopt the widely used *Independent Cascade* (IC) Model [KKT03a] to model the influence propagation. Specifically, we use a weighted directed graph to model the users and their relationships. A node in the graph represents a user. Each edge (u, v) is assigned with a propagation probability which ranges over $(0, 1]$. In the IC model, each node is either active or inactive and a node is only allowed to turn from inactive to active, but not vice versa. By assuming that a regional marketing strategy can directly affect the people who visit the region, these people will serve as the set of seeds S in the influence propagation following the IC model. Specifically, let S_t denote the set of users activated at time step t and $S_0 = S$. At time step $t + 1$, each user $v \in S_t$ has a single chance to activate each currently inactive neighbor u with a probability $p(v, u)$. The propagation process terminates when $S_t = \emptyset$. Thus, the influence of a region is the expected number of influenced users.

Application 2: Most Diversified Region. The second application of the *BRS* problem is to find the most diversified region. Specifically, given the size of a region, we can apply our algorithms to find a region with maximum diversity. We consider a set of spatial objects, each of which is associated with a set of tags to indicate its categories, *e.g.*, “restaurant” and “bar”. The *diversity* of a region is then measured by the number of different tags of the spatial objects inside the region. In this application, the submodular monotone function is $f(O_i) = |\bigcup_{o \in O_i} L(o)|$, for $O_i \subseteq O$, where O is the set of spatial objects, and $L(o)$ is the set of tags that are associated with spatial object o .

Table 3.2: Summary of datasets for Application 1.

Property	Brightkite	Gowalla
# of objects	693,362	1,256,692
# of check-ins	4,491,143	6,442,890
# of users	58,228	196,591
Width	314.391	495.658
Height	359.823	353.771

Table 3.3: Summary of datasets for Application 2.

Property	Yelp	Meetup
# of objects	48,753	589,715
# of tags/POI	48	14.7
Width	23.165	355.839
Height	123.936	180

Datasets To support evaluation of our algorithms for the first application, we use two real-life datasets², **Brightkite** and **Gowalla**, whose properties are given in Table 3.2. We

²<http://snap.stanford.edu/data/index.html>

use another two datasets, **Meetup** and **Yelp**³, to evaluate our algorithms for the second application. The properties of these two datasets are reported in Table 3.3.

Application 1 (Most Influential Region). **Brightkite** and **Gowalla** are from location-based social networks where users shared their locations by check-ins. Each dataset consists of a social graph, a set of points of interests, and a set of check-ins made by users. **Brightkite** contains 4,491,143 check-ins made in 693,362 places from 58,228 users. **Gowalla** contains 6,442,890 check-ins made in 1,256,692 places from 196,591 users. For each edge (u, v) in the social graph, we generate its propagation probability by a widely adopted random method [CWW10a, JHC12b, LCZ⁺14], *i.e.* the propagation probability of each edge is randomly selected from $\{0.1, 0.01, 0.001\}$. We compute the probability of a user u visiting a place p by $\frac{\# \text{ of check-ins in } p \text{ of } u}{\# \text{ of check-ins of } u}$, which is the ratio of the number of check-ins in p made by u to the number of check-ins made by u . We adopt the method [TXS14, BBCL14], referred to as the *Reverse Influence Sampling*, to approximate the influence spread of a set of users.

Application 2 (Most Diversified Region) **Meetup** is crawled by ourselves from an event-based social network *meetup.com* from July 2013 to Oct 2013. Each user can specify the topics that they are interested in and the website helps them to arrange a place to meet. Each spatial object represents a venue where people have held at least one event. For each venue, we select the most frequent topics of the users who have attended an event in this place and associate such topics with this venue. To generate the **Yelp** dataset, we collect the reviews for each POI. After removing the stop words and stemming, we collect the most frequent words in the reviews and let them be the tags of each POI.

Evaluated algorithms We evaluate the following algorithms. (a) *SliceBRS* algorithm; (b) *CoverBRS* algorithm with parameter $c = 1/3$, denoted as *CoverBRS4*; (c) *CoverBRS* algorithm with parameter $c = 1/2$, denoted as *CoverBRS9*, and (d) *Optimal Enclosure* algorithm proposed in [NB95] for addressing the *MaxRS* problem, denoted as *OE*. Recall that the *CoverBRS* algorithm returns an approximate answer to the *BRS* problem while *SliceBRS* finds the exact answer. In the *SliceBRS* algorithm, we need to partition the space into slices with fixed width. In our experiments, we set the width of each slice as b , where b is the width of the given rectangle. The *OE* algorithm is designed for the *MaxRS* problem and can be regarded as a heuristic algorithm for the *BRS* problem. Note that the aggregation of the objects in the region found by the *OE* algorithm for the *BRS* problem does not have an approximate bound to the optimal value. Due to space constraints, experiments related to the adaptation of *SliceBRS* algorithm to efficiently solve the *MaxRS* problem is reported in Section 3.4.7.

Query Rectangles As the cardinality of the datasets has an influence on the efficiency of the algorithms, it is not a good idea to use the same size of query rectangle on

³http://www.yelp.com.sg/dataset_challenge

 Figure 3.10: Quality vs. kq .

different datasets. Therefore, we consider the cardinality of the datasets by setting $q = \frac{Height}{|O|} \times \frac{Width}{|O|}$ as the unit size of query rectangle where *Height* and *Width* are the height and width of the minimum rectangular space that can include all spatial objects and $|O|$ is the number of spatial objects. Let $k \cdot q = (k \cdot \frac{Height}{|O|}) \times (k \cdot \frac{Width}{|O|})$. We vary the size of query rectangle by using different values for k .

Performance measures We consider the following performance measures: (a) The runtime of each algorithm and (b) the aggregate score of the spatial objects in rectangular region $r_p^{a,b}$, where p is the point returned by each algorithm and the aggregate score is denoted by $f(\cdot)$ in our problem definition. In the subsequent discussion, we refer to the latter measure as “quality” of the results.

3.4.2 Quality and Efficiency

We first conduct a set of experiments to evaluate the performance of our algorithms in the aforementioned applications. We use the following five sizes for query rectangles, q , $5q$, $10q$, $15q$, and $20q$.

Application 1 (Most Influential Region). We first study the performance of our algorithm for Application 1, *i.e.* finding the most influential region, in terms of both quality and runtime. Figure 3.10 reports the quality of the returned region for query rectangles of different sizes. First, we observe that the regions returned by *CoverBRS4* and *CoverBRS9* have comparable quality to the regions returned by *SliceBRS*. Second, the region returned by *OE* has the worst quality. This is because *OE* is designed for the *MaxRS* problem, which adopts different aggregate function from that used in the *BRS* problem. Thus, it is inappropriate to adopt *OE* to solve the *BRS* problem. In the subsequent discussions related to efficiency and scalability, we do not compare *OE* with our techniques.

 Figure 3.11: Runtime vs. kq .

Figure 3.11 depicts the runtime of our three algorithms for query rectangles of different sizes. Note that the y -axis is in logarithmic scale. Observe that although *SliceBRS* can complete in reasonable time, it is less efficient than *CoverBRS4* and *CoverBRS9*. This is due to the reasons mentioned in the preceding section: (1) the *CoverBRS* algorithm can efficiently select a c -cover T , (2) the c -cover T is sparser and has less spatial objects than the original set of spatial objects O , and (3) the new problem instance defined on T needs less computation.

Application 2 (Most Diversified Region). Next, we investigate the performance of our algorithm for Application 2. Figure 3.12 and Figure 3.13 report the quality and runtime of the evaluated algorithms, respectively. We can make similar observations as those on Figure 3.10 and Figure 3.11.

3.4.3 Usefulness of Optimization Strategies

We conduct several experiments to validate the usefulness of the various strategies we proposed in Sections 3.2 and 3.3 to improve efficiency of our proposed algorithms.

Usefulness of Maximal Region We first evaluate the usefulness of maximal region in improving the performance of our algorithm. We use $10q$ as the size for the query rectangle for each dataset. We compare (1) the number of disjoint regions, denoted by $\#DR$, and (2) the number of maximal regions, denoted by $\#MR$. The results are reported in Table 3.4. Observe that compared to the number of disjoint regions, the number of maximal regions is smaller. In fact, the number of maximal regions is about 1% of the number of disjoint regions. Thus, maximal regions enables us to reduce search space significantly and yield better efficiency.

Figure 3.12: Quality vs. kq .

Table 3.4: Effectiveness of maximal regions.

	Dataset	#DR	#MR
App. 1	Brightkite	38,757,062	247,332
	Gowalla	46,582,671	349,723
App. 2	Yelp	6,605,228	40,460
	Meetup	22,786,743	167,789

Usefulness of Maximal Slab Next, we investigate the usefulness of maximal slabs in improving the efficiency of our algorithm. By using $10q$ as the size for the query rectangle, we compare (1) the number of maximal regions, denoted by $\#MR$, (2) the number of maximal slabs, denoted by $\#MS$, (3) the number of maximal slabs that are processed by **SearchMR**, denoted by $\#MSP$, and (4) the number of disjoint regions (including maximal regions) that the algorithm actually process, denoted by $\#DRP$ (Note that it is possible that a non-maximal region is processed by our algorithm). The results are reported in Table 3.5. We can see that only a small part of the maximal slabs are processed by **SearchMR**. Furthermore, compared to the number of maximal regions, only a small set of disjoint regions is processed. Note that our maximal slab-based pruning technique has a better performance on **Yelp**, **Brightkite** and **Gowalla**. This is because that two venues in **Meetup** share many common tags, the upper bound that we estimate for a maximal slab is loose. Since the upper bounds of many maximal slabs are larger than the current result, we need to keep processing these maximal slabs by invoking **SearchMR**.

Usefulness of Cutting Space into Slices In this set of experiment, we evaluate the usefulness of the idea of cutting the space into slices. We implement an algorithm, namely *SliceBRS-NoSlice*, which is the *SliceBRS* algorithm without cutting the space into slices.

Figure 3.13: Runtime vs. kq .

Table 3.5: Effectiveness of maximal slabs.

	Dataset	#MR	#MS	#MSP	#DRP
App. 1	Brightkite	247,332	649,442	39	1,249
	Gowalla	349,723	1,206,367	132	8,417
App. 2	Yelp	40,460	33,699	170	6,342
	Meetup	167,789	516,379	3,257	199,975

We use 5 sizes of query rectangles, q , $5q$, $10q$, $15q$, and $20q$ on **Brightkite** and compare the runtime of *SliceBRS* and *SliceBRS-NoSlice*. Note that *SliceBRS-NoSlice* runs out of memory when the size of query rectangle is $20q$. Figure 3.14 reports the runtime of the two algorithms for 4 query rectangles of different sizes, q , $5q$, $10q$ and $15q$. Notice that *SliceBRS* is orders of magnitudes faster than *SliceBRS-NoSlice*. Thus, the idea of cutting the space into slices greatly improves the efficiency of the *SliceBRS* algorithm.

Usefulness of c -cover We now evaluate the usefulness of c -cover in improving the efficiency of our algorithms. We use $10q$ as the size for the query rectangle and compare the (1) the number of spatial objects in the original set of spatial objects O , denoted by $|O|$, (2) the number of spatial objects in the c -cover, denoted by $|T|$, (3) the number of disjoint regions in the new instance, denoted by $\#DR_T$, (4) the number of maximal regions in the new instance, denoted by $\#MR_T$, and (5) the number of disjoint regions that the algorithm actually processed, denoted by $\#DRP_T$. Table 3.6 reports the results. We can observe that the number of spatial objects in the new instance is smaller than in the original set of spatial objects. In addition, both the number of maximal regions and the number of disjoint regions that are processed are greatly reduced compared to the original instance. Therefore, the c -cover is very useful in efficiently finding an approximate answer to the *BRS* problem.

 Figure 3.14: Runtime of *SliceBRS* with and without cutting the space into slices.

 Table 3.6: Effectiveness of c -covers.

	Dataset	$ O $	$ T $	$\#DR_T$	$\#MR_T$	$\#DRP_T$
App. 1	Brightkite	693,362	91,308	223,769	39,156	8
	Gowalla	1,256,692	158,069	390,003	70,656	12
App. 2	Yelp	48,753	6,133	36,304	2,301	86
	Meetup	589,715	91,547	201,779	32,222	389

3.4.4 Effect of Slice Width

Next, we investigate the effect of slice width on our proposed algorithms.

The value for θ controls the width of each slice. In this experiment, we use $10q$ as the query and vary the value of θ from 1 to 5 with an increment of 1 for each dataset in the two applications.

Application 1 (Most Influential Region). Figure 3.15 plots the runtime of the three algorithms with different slice widths in Application 1. Observe that as the width increases, the runtime of *SliceBRS* increases. This is because that the space will be cut into lesser slices when the slice width is larger. Consequently, there are potentially more maximal regions in a slice and lesser maximal regions can be pruned. We can also observe that *CoverBRS4* and *CoverBRS9* are less sensitive to the slice width.

Application 2 (Most Diversified Region). The results for Application 2 are reported in Figure 3.16. We can make similar observations as those on Figure 3.15.

In summary, observe that θ can be set to 1 for the aforementioned applications. More importantly, there are potentially many maximal regions can be pruned and a little redundant computation since each rectangle will only be considered in at most two

Figure 3.15: Runtime vs. slice width in Application 1.

slices. This set of experiments proves again that the idea of cutting space into slices greatly improves the efficiency of *SliceBRS*.

3.4.5 Effect of the Query Rectangle

We study the effect of the query rectangle on the performance of our algorithms from two aspects: the size and the aspect ratio.

We first study the effect of the size of the query rectangle. We use five sizes for query rectangles, q , $5q$, $10q$, $15q$, and $20q$.

Application 1 (Most Influential Region). We first study the effect of the size for the query rectangle in Application 1. Figure 3.10 reports the quality of the returned region when we vary the size for the query rectangle in the Application 1. We can see that as the size for the query rectangle gets larger, the gap between the quality of the region returned by *SliceBRS* and *OE* gets smaller. Figure 3.11 reports the runtime of our three algorithms and the *OE* algorithm when we vary the size for query rectangle. We observe that as the size of the query rectangle gets larger, the runtime of *SliceBRS* gets larger. The runtime of *CoverBRS4* and *CoverBRS9* also increase as the size of the query rectangle gets larger, but at a slower rate.

Application 2 (Most Diversified Region). Figure 3.12 reports the quality of the returned region when we vary the size of query rectangles in the Application 2. Figure 3.13 presents the runtime of the algorithms. We can make similar observations as those on Figures 3.10 and 3.11.

Lastly, we study the effect of the aspect ratio of the query rectangle on **Gowalla**. We use 5 aspect ratios, 1:2, 2:3, 1:1, 3:2, 2:1, i.e., the query rectangle varies from wide

(a) Yelp

(b) Meetup

Figure 3.16: Runtime vs. slice width in Application 2.

Figure 3.17: Effect of Aspect Ratio.

Figure 3.18: Runtime vs. graph size.

rectangle to tall rectangle. Figure 3.17 reports the runtime of our three algorithms for different aspect ratios. We observe that, the runtime for a square query rectangle is slightly larger than those on other rectangles. To explain this, we consider a query rectangle of size $a \times b$ where a takes a very small value. Then based on the *SIRI* problem, the rectangles centered at each object can hardly intersect with each other, yielding nearly n maximal regions. Consequently, it is likely to take more time to process a square query rectangle.

3.4.6 Scalability with Graph Size

Lastly, we investigate the scalability of our three algorithms. We generate synthetic datasets under Gaussian distribution for Application 2. We set the number of spatial

Table 3.7: The ratio of *SliceBRS*'s runtime to *OE*'s runtime.

Query Rectangle	Brightkite	Gowalla	Yelp	Meetup
q	32.3%	29.9%	35.2%	33.5%
$5q$	22.9%	24.2%	32.1%	27.6%
$10q$	22.5%	25.1%	30%	22.4%
$15q$	23.4%	21.2%	35.2%	21.8%
$20q$	25.8%	23.9%	39.2%	22.2%

objects of dataset to be from 20,000,000 to 120,000,000 . By following the default setting in the work that is most germane to our work[CCT12], the spatial objects fall in a space of $10^9 \times 10^9$ and the size of the query rectangle is $10^6 \times 10^6$. For each spatial object, we randomly assign three labels from 388 categories in Foursquare. We report the runtime of the three algorithms in Figure 3.18. Note that the y -axis is in log-scale. From the figure we can observe that the two approximate algorithms scale well with the graph size. The exact algorithm becomes significantly slower when the number of spatial objects in the dataset gets larger.

3.4.7 Application to the MaxRS Problem

The *MaxRS* problem is a special case of the *BRS* problem. Hence, our proposed algorithm can also be applied to solve the *MaxRS* problem. In this set of experiments, we compare the runtime of *OE* and an adapted version of *SliceBRS* for the *MaxRS* problem. We show that our *SliceBRS* algorithm can also be adapted to efficiently solve the *MaxRS* problem. For each dataset, we find the region of given size such that the number of POIs that are inside the region is maximized. We adapt the *SliceBRS* algorithm to make use of the SUM aggregate score function as it is used in *OE*. Specifically, in each slice, we mark all maximal slabs whose upper bounds are higher than the current result. The rectangles which do not intersect with these marked maximal slabs can be safely ignored. Then, we modify Function **SearchMR** to incorporate the idea of *OE* to find the point p with maximum $h(A(p))$ in the remaining maximal slabs. In this case, the complexity of *SliceBRS* is $O(n \log n)$. Note that the modification to *SliceBRS* cannot work with a general submodular monotone aggregate function. Table 3.7 reports the ratio of the runtime of the adapted *SliceBRS* to that of *OE*. We can see that the time cost of the adapted *SliceBRS* is about 20% to 40% of *OE*. In other words, our *SliceBRS* algorithm can also be adapted to efficiently solve the *MaxRS* problem.

3.5 Summary

The quest for high quality location-based services has become more pressing due to explosive growth of mobile devices and geo-tagged data. In this chapter, we introduce the best region search (*BRS*) problem that aims to find a rectangular region with a given size such that the aggregation of the spatial objects in the region is maximized. This problem is fundamental to supporting several location-based applications such as most influential region search and most diversified region search. To this end, we propose a novel algorithm called *SliceBRS* that employs several pruning strategies to find the exact answer to the *BRS* problem. Since slight imprecision is acceptable in many real-world applications, we further propose the *CoverBRS* algorithm to find a constant-bounded answer to the *BRS* problem with a much better efficiency. The experimental study demonstrates that *SliceBRS* can find the exact answer to the *BRS* problem whereas *CoverBRS* can find a competitive answer with a better efficiency. These algorithms scale well with the number of spatial objects. As part of future work, we intend to explore efficient techniques to find *top-k regions* in the context of the *BRS* problem and explore *BRS* problem in the context of road networks.

Chapter 4

Bursty Region Detection

This chapter proposes a novel *continuous bursty region detection* (SURGE) problem that utilizes such spatial object stream to detect and maintain *bursty region* of a given size in a specified geographic area in real time. Specifically, a bursty region shows maximum spike in the number of spatial objects in a given time window. An exact solution and two approximate solutions are proposed. Specifically, the approximate solutions can monitor a region whose *burst score* is bounded by a $\frac{1-\alpha}{4}$, where α is a parameter to control the burst score. We further extend these solutions to support detection of top- k bursty regions. Extensive experiments with real-world data are conducted to demonstrate the efficiency and effectiveness of our solutions.

This chapter is organized as follows. We formally introduces the SURGE problem in Section 4.1 . We then present the exact and approximate solutions to the SURGE problem in Sections 4.2 and 4.3, and discuss the extensions of our proposed solutions to detect top- k bursty regions in Section 4.4. The experimental study is presented in Section 4.5, followed by summary in Section 4.6. We summarize notations used in Table 4.1.

4.1 Problem Statement

We formally define the *ContinuousS BUrsty ReGion DEtection* (SURGE) problem. We begin by defining some terminology.

4.1.1 Terminology

A *spatial object* is represented with a triple $o = \langle w, \rho, t_c \rangle$, where w is the weight of o , ρ is a location point with latitude and longitude, and t_c is the creation time of object o . In this chapter, we consider a stream of spatial objects. For example, geo-tagged tweets in *Twitter* can be viewed as a stream of spatial objects arriving in the order of creation time. The *weight* of a tweet could be the relevance of its textual content to a set of query

Table 4.1: Table of notations

Notation	Definition
$\mathcal{O}, \mathcal{G}$	The stream of spatial objects and rectangle objects
W_c, W_p	Current time window, past time window
$\mathcal{B}, S_c, S_p$	Burst score, Current score, past score
$a \times b$	The size of the query rectangle
$\mathcal{O}, \mathcal{O}_r$	The set of spatial objects, the set of spatial objects in r
$\mathcal{G}, \mathcal{G}_p$	The set of rectangle objects, the set of rectangle objects that can cover p
I	An interval
o, g, p	A spatial object, a rectangle object, a point
r, c	A region r , a cell c
$\bar{c}_{cold}, \bar{c}_{hot}, \bar{c}$	The cold upper bound, hot upper bound, and upper bound for a cell.

keywords. The car requests in *Uber* can also be viewed as a stream of spatial objects arriving in the order of calling time. In this case, the weight could be the passenger number or travel fare.

We next introduce two consecutive time-based sliding windows, namely *current* and *past windows*. Given a window size $|W|$, the *current window*, denoted by W_c is a time period of length $|W|$ that stretches back to a time point $t - |W|$ from present time t . The *past window*, denoted by W_p is a time period of length $|W|$ that stretches back to a time point $t - 2|W|$ from the time point $t - |W|$.

Given a region r and a sliding window W , let $O(r, W)$ be the set of spatial objects which is created in W and located in region r , i.e., $O(r, W) = \{o | o.p \in r \wedge o.t_c \in W\}$. Let $f(r, W)$ be the summation of weights of objects in $O(r, W)$ normalized by W 's length, i.e., $f(r, W) = \frac{\sum_{o \in O(r, W)} o.w}{|W|}$, which is the score of a region r w.r.t. the sliding time window W .

Note that in this work, for the sake of simplicity, we assume the current window and the past window have the same length $|W|$. However, our proposed solution is equally applicable when the two sliding windows have different lengths.

4.1.2 Burst Score

Intuitively, the *burst score* of a region r reflects the variation in the spatial objects in r in recent period. This motivates us to design the burst score based on the current and past windows.

We first discuss the intuition in designing the *burst score* using the Example 4 in Chapter 1.

In this scenario, *Uber* drivers are interested in regions in which they have a higher chance to pick up a passenger. Obviously, a driver is more likely to find a passenger in a

region that contains a large number of requests in the current window, which represents the *significance* of the region. On the other hand, if a region suddenly experiences a surge of requests, which represents the *burstiness* of the region, then it is highly likely that existing drivers in that region may not be able to fulfill this sudden increase in demand. Consequently, a driver will have a higher chance to find a passenger there.

Thus, we consider the following two factors in our burst score: (a) The score of the region w.r.t. the current window, i.e. $f(r, W_c)$, which measures the *significance*, and (b) the increase in the score of the region between the current window and the past window, i.e., $\max(f(r, W_c) - f(r, W_p), 0)$, which measures the *burstiness*. Note that we use the *max* function to guarantee that the increase in the score between the current and past windows is always non-negative since we are only interested in increase in the score.

We now formally define the burst score as follows.

Definition 9: Burst Score. Given a region r , we define its burst score $\mathcal{S}(r)$ as:

$$\mathcal{S}(r) = \alpha \max(f(r, W_c) - f(r, W_p), 0) + (1 - \alpha)f(r, W_c), \quad (4.1)$$

where $\alpha \in [0, 1)$ is a parameter that balances the significance and the burstiness. $\square$

4.1.3 Continuous Bursty Region Detection (SURGE) Problem

We are now ready to formally define the SURGE problem.

Definition 10: Continuous Bursty Region Detection (surge) Problem. Consider a stream of spatial objects $\mathcal{O}$. Let $q = \langle A, a \times b, |W| \rangle$ be a SURGE query where A is a preferred area, $a \times b$ is the size of the query rectangle, and $|W|$ is the length of the current and past windows. Given such a query q , the aim of the **surge problem** is to continuously detect the position of the region r of size $a \times b$ in A with the maximum burst score. The region r is referred to as the **bursty region**. $\square$

4.2 An Exact Solution

The SURGE problem is challenging to address due to the following reasons. First, given a snapshot of the stream, we are required to locate the bursty region in the preferred area A . Intuitively, this bursty region can be located at any point and it is prohibitively expensive to check the region located at every point, which is infinite. Second, whenever a spatial object enters or leaves the sliding windows, the burst score of any region which encloses this object will change. This implies that the location of the bursty region may change as well and we need to recompute the new bursty region. With the high arrival rate of the stream, it demands an efficient strategy to update the bursty region.

In this section, we present a solution to address the SURGE problem. We first introduce the *continuous bursty point detection* (cSPOT) problem in Section 4.2.1. We show that by reducing the SURGE problem to the cSPOT problem, for any snapshot of the stream, we convert the challenge of selecting a point from infinite points in the preferred area A to selecting a **bursty point** from $O(n^2)$ disjoint regions. To address the second challenge, we present a cell-based algorithm to continuously update the **bursty point** in Section 4.2.3.

4.2.1 The cSPOT Problem

We next define the cSPOT problem and present how to reduce the SURGE problem to the cSPOT problem. Firstly, we introduce some terminology that will be used to define the cSPOT problem.

Definition 11:Rectangle Object. A rectangle object, denoted with a triple $g = \langle w, \rho, t_c \rangle$, is a rectangle of size $a \times b$, where $g.w$ is its weight, $g.\rho$ is the location of its left-bottom corner, and $g.t_c$ is the creation time of g . $\square$

Given the stream of spatial objects $\mathcal{O}$, each spatial object o in $\mathcal{O}$ can be mapped to a rectangle object g by using o as the left-bottom corner, i.e., $g.w = o.w$, $g.\rho = o.\rho$, and $g.t_c = o.t_c$. Let $\mathcal{G}$ denote the stream of rectangle objects that are mapped from $\mathcal{O}$. Let $G(p, W)$ be the set of rectangle objects which covers point p and is created in window W , i.e., $G(p, W) = \{g | g.t_c \in W \wedge p \in g \wedge g \in \mathcal{G}\}$.

Next, we define the burst score of a point by following the definition of burst score of a region in Section 4.1. With a slight abuse of notation, we continue to use $f(p, W)$ and $\mathcal{S}(p)$ to denote the score of a point p w.r.t. the window W , and the burst score of p , respectively.

Definition 12:Burst Score of a Point. Consider a stream of rectangle objects $\mathcal{G}$. The burst score $\mathcal{S}(p)$ of point p is defined as

$$\mathcal{S}(p) = \alpha \max(f(p, W_c) - f(p, W_p), 0) + (1 - \alpha)f(p, W_c)$$

where W_c and W_p are the current and past windows, and for a sliding window W , score $f(p, W)$ is the summation of weights of rectangle objects in $G(p, W)$, i.e., $f(p, W) = \frac{\sum_{g \in G(p, W)} g.w}{|W|}$, which is the score of a point p w.r.t. the sliding time window W . $\square$

We are now ready to formally define the cSPOT problem.

Definition 13:cSPOT Problem. Consider a stream of rectangle objects $\mathcal{G}$, a parameter α , as well as the current window W_c and past window W_p . The *Continuous Bursty Point Detection* (cSPOT) problem aims to keep track of a point p in the space, such that its

Figure 4.1: Reduce to cSPOT problem

burst score $\mathcal{S}(p)$ is maximized. A point p with the maximum score is referred to as **bursty point**. $\square$

In order to reduce the SURGE problem to the cSPOT problem, for each spatial object o in the SURGE problem, if o is in the preferred area A , i.e., $o.p \in A$, we generate a rectangle object g of size $a \times b$ with o as the left-bottom corner such that $o.t_c = g.t_c$ and $g.p = o.p$. We illustrate this reduction with the example in Figure 4.1. Assume that $o_1, \dots, o_3$ are all in A . For each spatial object $o_i, i \in [1, 3]$, a corresponding rectangle object g_i is generated. We next show the relationship between the bursty region and the bursty point of the corresponding SURGE and cSPOT problem.

Theorem 8: Let p_m be a bursty point for the reduced cSPOT problem given a snapshot. The rectangular region r_m of size $a \times b$ whose top-right corner is located at p_m is a bursty region for the original SURGE problem for the snapshot. $\square$

Proof: Let p be any point in the cSPOT problem, and r be the rectangular region of size $a \times b$ whose top-right corner is located at p . A spatial object o is in r iff the corresponding rectangle object g can cover p . Since the corresponding o and g have the same creation time and weight, we can derive that $f(r, W_c) = f(p, W_c)$, $f(r, W_p) = f(p, W_p)$, and thus r and p have the same burst score. As a result, if the point p_m has the maximum burst score in the cSPOT problem, then r_m also has the maximum burst score in the SURGE problem.

Note that the reduction is inspired by the idea of transforming the max-enclosing rectangle problem to the *rectangle intersection problem* [NB95]. The *rectangle intersection problem* aims to find the most overlapped area given a set of rectangles. Since our problem has a different burst score function, the techniques designed for the rectangle intersection problem cannot be utilized to search for the bursty point at a snapshot.

We address the SURGE problem by solving the corresponding cSPOT problem. Observe that in the cSPOT problem, the edges of the rectangle objects divide the space into many disjoint regions. Consider the example in Figure 4.1. The shaded area is one of

the disjoint region which is the overlap of g_1 , g_2 , and g_3 . All points in a disjoint area are covered by the same set of rectangles. Thus they have the same burst score. Next we present a theorem which justifies the reason behind the reduction.

Theorem 9: Given a snapshot of the stream of rectangle objects in the CSPOT problem, there are at most $O(n^2)$ disjoint regions, where n is the number of rectangle objects in windows W_c and W_p . [NB95] $\square$

Since all points in a disjoint region have the same burst score, Theorem 9 tells us that we only need to consider $O(n^2)$ disjoint regions, which addresses the first challenge of the SURGE problem, i.e., locating the bursty region from infinite possible locations.

Example 16: Consider a snapshot of the stream shown in Figure 4.1. Assume that o_1 , o_2 and o_3 are three spatial objects in the current window W_c in the SURGE problem, and $o_i.w = 1$ for $i \in [1, 3]$. According to the reduction process, g_1 , g_2 and g_3 are three rectangle objects in the current window in the CSPOT problem, and $g_i.w = 1$ for $i \in [1, 3]$. Assume that $|W_c| = 1$. The shaded area is the intersection of g_1 , g_2 and g_3 . Thus, any point p in the shade area has the maximum burst score, i.e., $\mathcal{S}(p) = 3$. The point p in the figure is a bursty point at the given snapshot. The solid line rectangle, whose top-right corner lies in p , is the bursty region as it encloses three spatial objects and its burst score is 3. $\square$

We next present an exact solution to address the CSPOT problem efficiently. Specifically, given the stream of rectangle objects, we use a grid to divide the space into cells, and maintain the upper bounds of burst score for the points in each cell. Several optimization techniques are proposed to avoid redundant recomputation. If the upper bound of any cell is larger than the score of the current bursty point, we invoke a sweep-line based algorithm to search the cell to update the location of the bursty point.

In the rest of this section, we first introduce the sweep-line based algorithm, which finds the bursty point given a set of rectangle objects (Section 4.2.2). Then we present the cell-based lazy update strategy, which determines whether we should invoke the sweep-line algorithm to recompute the bursty point (Section 4.2.3).

4.2.2 Detecting Bursty Point on a Snapshot

To address the first challenge, i.e., detecting the bursty point given a snapshot of the stream, we propose a sweep-line based algorithm called SL-CSPOT in this subsection.

The high level idea of the SL-CSPOT algorithm is as follows. We use a horizontal line, referred to as the sweep-line, to scan the space top-down. The sweep-line is divided into $2n + 1$ intervals at most by the vertical edges of the n rectangle objects. For instance, in Figure 4.2, the vertical edges of the three rectangles divide the sweep-line into 7 intervals, $\{I_0, \dots, I_6\}$. For each interval I , we use $I.f_c$ and $I.f_p$ to denote the score w.r.t. the current

Algorithm 3: SL-CSPOT Algorithm**Input:** A set of rectangle objects G **Output:** A bursty point p

```

1  $p = null$ ;
2 while sweep-line meets an horizontal edge of a rectangle  $g$  do
3    $I_i, \dots, I_j \leftarrow$  the intervals covered by  $g$ ;
4   for interval  $I \in \{I_i, \dots, I_j\}$  do
5     Update  $I.f_c$ ,  $I.f_p$  and  $I.S$ ;
6     if  $I.S > S(p)$  then
7        $p \leftarrow$  a point beneath  $I$ , and between the sweep-line and next horizontal edge;
8 return  $p$ ;

```

and past windows, respectively for the points on the interval I . We use $I.S$ to denote the burst score of such points. For any interval I_i , the set of rectangles which can cover interval I_i changes when the sweep line meets the top or bottom edge of a rectangle which can cover I_i , and its burst score $I_i.S$ is updated accordingly. A point with the maximum burst score during the sweeping process is returned as the bursty points.

We next illustrate the algorithm with an example. Figure 4.2 shows a snapshot of the stream. Rectangle g_1 is in the past window W_p (marked in blue), while g_2 and g_3 are in the current window W_c (marked in red). As shown in Figure 4.2, when the sweep-line meets the top edge of g_3 , any point, such as p_1 , which is beneath the overlapped intervals I_3 , I_4 and I_5 and above the next horizontal line, will be covered by g_3 . Since g_3 is in the current window, the score of p_1 w.r.t. W_c will be increased by $\frac{g_3.w}{|W_c|} = 2$, resulting in an increase of its burst score. We set $I_i.f_c = 2$ and $I_i.f_p = 0$ for $i \in [3, 5]$, and thus $I_i.S = 0.5 \cdot \max(I_i.f_c - I_i.f_p, 0) + 0.5 \cdot I_i.f_c = 2$ for $i \in [3, 5]$. We select p_1 as the current bursty point. Then the sweep-line meets the top edge of g_1 and g_2 , consecutively. The two edges are processed similarly, and we have $I_4.S = 3$. Thus p_3 is selected as the new bursty point. When the sweep-line meets the bottom edge of the rectangle g_3 , any point, such as p_4 , which is beneath the overlapped intervals and above the next horizontal line, will no longer be covered by g_3 . Thus, the scores w.r.t. W_c of the overlapped intervals $I_3, \dots, I_5$ are decreased. We have $I_i.f_c = 1$ for $i \in [3, 4]$, and $I_5.f_c = 0$. Their burst scores are updated as: $I_3.S = 1 - \alpha$, $I_4.S = 1$ and $I_5.S = 0$. We repeat this process until the whole space is scanned. Point p_3 has the maximum burst score during the sweeping process. Thus p_3 is returned as the bursty point.

Algorithm 3 outlines this procedure. It takes as input a set of rectangle objects G , and outputs a bursty point p with the maximum burst score in the space. Result point p is initialized as *null*. The algorithm uses a sweep-line to scan the space (lines 2–7). When it meets an horizontal edge of a rectangle r , it first locates the intervals that are covered by r (line 3). Then it updates $I.S$ for each interval I one by one (line 5). The point p is updated if any interval has a larger burst score (lines 6–7).

Figure 4.2: Illustration of bursty point detection.

Time Complexity. Let n be the number of rectangles in the space. The sweep-line scans $2 \cdot n$ edges (each rectangle has two horizontal edges). In the worse case, when the sweep-line meets an horizontal edge, $2 \cdot n + 1$ intervals are all affected. As a result, the time complexity of Algorithm 3 is $O(n^2)$.

4.2.3 Handling the Stream

We have presented Algorithm SL-CSPOT to detect a bursty point given a snapshot of the stream. But how to continuously detect the bursty point? Recall that the burst score of a point is determined by the set of rectangle objects that cover it. The bursty point is likely to change when a rectangle object enters or leaves the sliding windows. Specifically, any of the following events may change the bursty point: (1) a new rectangle object enters the current window, (2) an existing rectangle object leaves the current window and enters the past window, and (3) an existing rectangle object leaves the past window. We refer to these three events as a *new event*, a *grown event*, and an *expired event*, respectively. We use a tuple $e = \langle g, l \rangle$ to denote an event, where g is the rectangle object, and l is one status from $\{New, Grown, Expired\}$ to indicate the type of the event.

Intuitively, a naïve idea is whenever an event happens, we invoke Algorithm 3 to detect a bursty point on the snapshot of the stream. However, this idea does not address the CSPOT problem efficiently. First, it is not necessary to search the whole space. When an event happens, it only affects the burst score of the points inside the rectangle object of the event. Second, frequent recomputation of the bursty point is computationally expensive. To address the two issues, we next present a cell-based algorithm called *Cell-CSPOT*.

Algorithm 4: *Cell-cSPOT* Algorithm

Input: An event $e = \langle g, l \rangle$
Output: A bursty point

- 1 $C_g \leftarrow$ cells that are overlapped with g ;
- 2 **for** $c \in C_g$ **do**
- 3 Update $U(c)$ using Eqn 4.2, 4.3, and status of $c.p$ using Lemma 17;
- 4 $c \leftarrow \arg \max U(c)$;
- 5 **while** $c.p$ is *invalid* **do**
- 6 $c.p \leftarrow \text{SL-cSPOT}(c)$;
- 7 $U_d(c) = \mathcal{S}(c.p)$;
- 8 $c \leftarrow \arg \max U(c)$;
- 9 **return** $c.p$

4.2.3.1 Cell-based Lazy Update

An event only affects the burst scores of the points inside the rectangle of the event. This locality property motivates us to divide the space into *cells*, and develop approaches to handle the cells that are affected by an event. We first define the *grid* that we use as follows.

Definition 14:Grid and Cell. We consider a grid as a set of vertical and horizontal lines defined by $x = i \cdot b, y = i \cdot a$ for all integers $i \in [-\infty, +\infty]$. For each cell c , we maintain a list of rectangle objects which overlap with the cell over the two sliding time windows W_c and W_p , denoted by $c.G$. $\square$

We have the following lemma based on obvious observations.

Lemma 14: A rectangle object of size $a \times b$ overlaps with at most four cells of the grid in Definition 14. $\square$

For each cell in the grid, we maintain a burst score upper bound for the points inside the cell (to be discussed in Section 4.2.3.2). When an event happens, the corresponding rectangle can only affect at most four cells. Instead of searching the affected cells immediately after an event happens, we propose a *lazy update strategy* by utilizing the maintained upper bound: Whenever an event happens, we first update the upper bounds of the affected cells. Then, we invoke Algorithm 3 to search the cells iteratively in the descending order of their upper bounds. In each iteration, we always search the cell with the maximum upper bound. We terminate the process when there is no upper bound larger than the current maximum burst score. Hence, when an event happens, if the upper bounds of the affected cells are less than the current maximum burst score, these cells will not be searched. Thus the lazy update strategy significantly reduces the number of times that Algorithm 3 is invoked to search affected cells.

In addition, to reuse the result of Algorithm 3 from previous iterations, we record the point returned by Algorithm 3 for each cell which is called *candidate point*. The status of each candidate point is either *valid* or *invalid*. If the candidate point of a cell is guaranteed to have the maximum burst score in the cell, its status is *valid*. On the other hand, the status is set to *invalid* if it is unknown whether the candidate point has the maximum burst score. We do not need to invoke Algorithm 3 to search a cell if its candidate point is *valid*. By exploiting the candidate points, we can further avoid searching in some cells (discussed in Section 4.2.3.3).

Algorithm 4 presents an overview of our algorithm called *Cell-cSPOT* (*cell*-based cSPOT). It takes as input an event $e = \langle g, l \rangle$, and reports a bursty point in the space. The algorithm first locates the set C_g of cells that overlap with g (line 1). Then for each cell c in C_g , it updates its upper bound based on Equations 4.2, and 4.3 (to be introduced in Section 4.2.3.2), and determine the status of the candidate point $c.p$ based on Lemma 17 (to be introduced in Section 4.2.3.3) (line 3). Then it accesses the cells in descending order of their upper bounds $U(c)$ iteratively (lines 4–8). In each iteration, if the candidate point $c.p$ is *invalid*, we invoke Algorithm 3 to search the cell and update $c.p$ (line 6) and the upper bound (line 7). Otherwise $c.p$ is *valid*, and this indicates that $c.p$ has the maximum burst score in cell c and c has the maximum burst score as there is no cell whose upper bound is larger than the current maximum burst score. Therefore we terminate the process and report point $c.p$ as the result.

Time Complexity. According to Lemma 14, at most four cells are affected by an event rectangle g . Thus, it takes $O(1)$ time to update the upper bounds and candidate points. A cell will not be searched unless it is overlapped with a rectangle object. Thus, $O(1)$ cells are searched in processing a rectangle object. In our implementation, we use a heap to maintain the cells based on their upper bounds. Let $|c_{max}|$ be the maximum number of rectangle objects in a cell. Let n be the number of rectangle objects created in W_c and W_p . It takes $O(\log n)$ time to get the cell c and $O(|c_{max}|^2)$ time to search the cell. Putting these together, the complexity of Algorithm 4 is $O(|c_{max}|^2 + \log n)$.

Space Complexity. Each rectangle object is stored in at most four cells. Thus, the space cost of Algorithm 4 is $O(n)$.

4.2.3.2 Upper Bound Estimation

Next, we present the details about estimating the upper bound for a cell.

Static Upper Bound. We first consider a simple strategy to estimate an upper bound for a cell. According to the definition of the burst score, rectangle objects in the current window have a positive impact on the burst score, while the rectangle objects in the past window have a non-positive impact. Hence, we can estimate an upper bound burst score

for a cell by only utilizing the objects in the current window. We refer to this upper bound as *static upper bound*.

Definition 15: Static Upper Bound. For a cell c , its static upper bound is computed as follows:

$$U_s(c) = \sum_{g \in c.G \wedge g.t_c \in W_c} \frac{g.w}{|W_c|} \quad (4.2)$$

where $c.G$ is a set of rectangle objects overlapped with c . □

Next, we show the correctness of the static upper bound.

Lemma 15: For any point p in a cell c , we have $\mathcal{S}(p) \leq U_s(c)$. □

Proof: We have

$$\begin{aligned} \mathcal{S}(p) &= \alpha \max(f(p, W_c) - f(p, W_p), 0) + (1 - \alpha)f(p, W_c) \\ &\leq \alpha f(p, W_c) + (1 - \alpha)f(p, W_c) = f(p, W_c) = U_s(c) \end{aligned}$$

Example 17: Consider the example shown in Figure 4.3. The solid-line rectangle is a cell in the grid. After event e_1 happens, there are three new rectangle objects overlapped with the cell c . The static upper bound of cell c is $U_s(c) = 3$. □

Dynamic Upper Bound. Next, instead of just using objects in the current window, we introduce another way to estimate the upper bound by using both the event and information from the previous computation. Specifically, when an event happens, we dynamically update the upper bound computed from previous upper bound. We refer to such upper bound as *dynamic upper bound*.

Let p_m be the point with the maximum burst score in cell c at a snapshot i when event e_i arrives. Apparently $\mathcal{S}(p_m)$ is an upper bound burst score for cell c at snapshot i . Thus, whenever we search a cell c with Algorithm 3 on a snapshot i , the dynamic upper bound $U_d^i(c)$ can be set as $U_d^i(c) = \mathcal{S}(p_m)$.

Let $U_d^i(c)$ be the upper bound of cell c on snapshot i when event e_i arrives, and $U_d^{i+1}(c)$ be the upper bound when e_{i+1} arrives. Let g be the corresponding rectangle object of e_{i+1} , i.e., $e_{i+1} = \langle g, l \rangle$. Then we have

$$U_d^{i+1}(c) = \begin{cases} U_d^i(c) + \frac{g.w}{|W_c|} & e_{i+1}.l \text{ is New,} \\ U_d^i(c) & e_{i+1}.l \text{ is Grown,} \\ U_d^i(c) + \alpha \frac{g.w}{|W_p|} & e_{i+1}.l \text{ is Expired} \end{cases} \quad (4.3)$$

We next show the correctness of the dynamic upper bound with the following lemma.

Lemma 16: Consider a cell c . For any point p in c , we have $\mathcal{S}(p) \leq U_d(c)$ after e happens. □

Figure 4.3: Cell upper bound.

Proof:

Let $\Delta\mathcal{S}(p)$, $\Delta f(p, W)$ be the increase of $\mathcal{S}(p)$ and $f(p, W)$ after e happens, respectively. We discuss the following three cases.

Case 1: e is new. For any point p that is covered by g , its current score is increased by $\Delta f(p, W_c) = \frac{g.w}{|W_c|}$. We have $\Delta\mathcal{S}(p) \leq \Delta f(p, W_c) = \frac{g.w}{|W_c|}$.

Case 2: e is grown. For any point p that is covered by g , its current score is decreased, i.e., $\Delta f(p, W_c) = -\frac{g.w}{|W_c|}$, and its past score is increased, i.e., $\Delta f(p, W_p) = \frac{g.w}{|W_p|}$. Thus, we can easily get $\Delta\mathcal{S}(p) \leq 0$.

Case 3: e is expired. For any point p covered by g , its current score is not affected, and its past score is decreased, i.e., $\Delta f(p, W_p) = -\frac{g.w}{|W_p|}$. Thus, we have $\Delta\mathcal{S}(p) \leq \alpha(-\Delta f(p, W_p)) = \alpha \frac{g.w}{|W_p|}$.

Since $\Delta\mathcal{S}(p) \leq \Delta U_d(c)$, we still have $\mathcal{S}(p) \leq U_d(c)$.

Example 18: Consider the example shown in Figure 4.3. We first consider an event $e_1 = \langle g_3, New \rangle$, i.e., a new rectangle object enters the current window. Assume before e_1 happens, we have searched the cell and the point p_1 has the maximum burst score in c . The dynamic upper bound is set as $U_d^0(c) = 1$. After e_1 happens, we update the dynamic upper bound as $U_d^1(c) = U_d^0(c) + \frac{g_3.w}{|W_c|} = 2$. Then we consider an event $e_2 = \langle g_1, Grown \rangle$, i.e., an existing rectangle object g_1 exits the current window and enters the past window. According to Eqn 4.3, the dynamic upper bound remains the same, i.e., $U_d^2(c) = 2$, since p_2 remains to have the maximum burst score in cell c . $\square$

We have presented the static upper bound and the dynamic upper bound. We now combine them for a tighter upper bound.

Definition 16: Upper bound for cell. For a cell c , we define its upper bound $U(c)$ as $U(c) = \min(U_s(c), U_d(c))$. $\square$

4.2.3.3 Candidate Point Maintenance

An expensive operation of Algorithm 4 is to invoke Algorithm 3 to find a point with the maximum burst score for a cell. To reuse the computation, for each cell c , we maintain a candidate point, denoted by $c.p$, to record the point returned by Algorithm 3. The candidate point has two possible status as introduced in Section 4.2.3.1. We next present Lemma 17, which is employed to determine the status of a candidate point.

Lemma 17: Let $c.p$ be a point with the maximum burst score in cell c currently. Consider an event $e = \langle g, l \rangle$. After e happens, if either (1) e is either new or expired, g can cover $c.p$, and $f(c.p, W_c) - f(c.p, W_p) > 0$, or (2) e is grown object and g cannot cover $c.p$, then the point $c.p$ still has the maximum burst score. $\square$

Proof: We use Δ to denote the increase of the score. We consider the following three cases.

Case 1: $e.l$ is new. We have $\Delta\mathcal{S}(c.p) = \frac{g.w}{|W_c|}$ if and only if g can cover $c.p$ and $f(c.p, W_c) - f(c.p, W_p) > 0$. In this case, $c.p$ still has the maximum burst score as $\Delta\mathcal{S}(p) \leq \frac{g.w}{|W_c|}$ for any p in g (Lemma 16). Otherwise, it is possible that there exists a point p' in g with a larger increase such that p' has a larger burst score than $c.p$ after g arrives.

Case 2: $e.l$ is grown. For any point p in g , the increase $\Delta\mathcal{S}(p) < 0$. If g does not cover $c.p$, $c.p$'s burst score does not change and it still has the maximum burst score. Otherwise, $c.p$'s burst score is decreased and could be exceeded by a point outside g .

Case 3: $e.l$ is expired. As shown in the proof for Lemma 16, $\Delta\mathcal{S}(p) \leq \alpha \frac{g.w}{|W_c|}$ for any p in g . We have $\Delta\mathcal{S}(c.p) = \alpha \frac{g.w}{|W_c|}$ if and only if g can cover $c.p$ and $f(c.p, W_c) - f(c.p, W_p) > 0$. In this case, $c.p$ still has the maximum burst score. Otherwise, similar to Case 1, it is possible that there exists a point p' in g with a larger increase of burst score.

Putting these together, the lemma is proved.

We determine the status of a candidate point based on Lemma 17. Consider a cell c and an event e which can affect c . If $c.p$ is valid and the conditions in Lemma 17 hold, then $c.p$ remains to be valid. Otherwise, $c.p$ is invalid after e happens.

Example 19: Reconsider the example shown in Figure 4.3. We consider the event $e_1 = \langle g_3, \text{New} \rangle$, where a new rectangle g_3 arrives. Before e_1 happens, assume that we have invoked Algorithm 3 to search the cell and p_1 is the point with the maximum burst score. When e_1 happens, since e_1 is new and g_3 cannot cover p_1 , p_1 is invalid after e_1 happens. In fact, points in the shaded area have the maximum burst score after e_1 happens. $\square$

Algorithm 5: GAP-SURGE Algorithm

Input: An event $e = \langle o, l \rangle$ **Output:** A cell c

```

1  $c_{i,j} \leftarrow$  the cell  $o$  lies in;
2 if  $e$  is new then  $c_{i,j}.f_c + = \frac{o.w}{|W_c|}$ ;
3 else if  $e$  is grown then  $c_{i,j}.f_c - = \frac{o.w}{|W_c|}$ ,  $c_{i,j}.f_p + = \frac{o.w}{|W_p|}$ ;
4 else  $c_{i,j}.f_p - = \frac{o.w}{|W_c|}$ ;
5  $c_{i,j}.S = \max(c_{i,j}.f_c - c_{i,j}.f_p, 0) + c_{i,j}.f_c$ ;
6  $c \leftarrow \arg \max c.S$ ;
7 return  $c$ 

```

4.3 Approximate Solutions

Although our exact solution can continuously detect the bursty region efficiently in real time, we observe that its runtime performance degrades when the number of spatial objects created in time windows W_c and W_p increases significantly (e.g., the sliding windows get larger, the region size gets larger, or the arrival rate of the spatial objects increases). Since a slight imprecision is acceptable in most cases in real life, to tackle this challenge, we propose two algorithms to solve the SURGE problem approximately. We prove that the burst score of the region returned by our proposed approximate algorithms is always bounded by a ratio $\frac{1-\alpha}{4}$ compared to the exact result.

4.3.1 A Grid-based Solution

The key idea behind our *grid-based approximate solution* is as follows: We use a grid to divide the space into cells of size $a \times b$. Each cell is a candidate region. By maintaining the burst score for each cell, we continuously report the cell with the maximum burst score to users as an approximation to the bursty region. A nice feature of this idea is that it is intuitive while it has performance guarantees.

Algorithm 5 outlines our proposed algorithm called GAP-SURGE (**G**rid-based **A**Pproximate **S**URGE). Here we abuse the notation $e = \langle o, l \rangle$ to denote an event of spatial object o enters or leaves the sliding windows. It first locates the cell that the spatial object o lies in (line 1). The burst score of the cell c is updated accordingly (lines 2–5). The cell with the maximum burst score is returned as an approximate result (line 6).

Before we show that the region returned by Algorithm 5 has a burst score with an approximation guarantee, we present some interesting properties of the burst score function.

Lemma 18: For any two region r_1 and r_2 , $r_1 \subseteq r_2$, we have $\mathcal{S}(r_2) \geq (1 - \alpha)\mathcal{S}(r_1)$. $\square$

Proof: According to the definition of the burst score, we have

$$\begin{aligned}\mathcal{S}(r_2) &= \alpha \max(f(r_2, W_c) - f(r_2, W_p), 0) + (1 - \alpha)f(r_2, W_c) \\ &\geq (1 - \alpha)f(r_2, W_c) \geq (1 - \alpha)f(r_1, W_c) \geq (1 - \alpha)\mathcal{S}(r_1)\end{aligned}$$

Lemma 19: Let r_1, r_2 be two non-overlapping regions. We have $\mathcal{S}(r_1) + \mathcal{S}(r_2) \geq \mathcal{S}(r_1 \cup r_2)$. $\square$

Proof: Since r_1 and r_2 are non-overlapping, according to the definition of burst score, we have

$$\begin{aligned}f(r_1, W_c) + f(r_2, W_c) &= f(r_1 \cup r_2, W_c) \\ f(r_1, W_p) + f(r_2, W_p) &= f(r_1 \cup r_2, W_p)\end{aligned}$$

Then we can easily get

$$\begin{aligned}&\max(f(r_1 \cup r_2, W_c) - f(r_1 \cup r_2, W_p), 0) \\ &\leq \max(f(r_1, W_c) - f(r_1, W_p), 0) + \\ &\quad \max(f(r_2, W_c) - f(r_2, W_p), 0)\end{aligned}$$

Thus, we have $\mathcal{S}(r_1 \cup r_2) \leq \mathcal{S}(r_1) + \mathcal{S}(r_2)$.

Now we are ready to prove the approximate ratio of Algorithm 5.

Theorem 10: Given a snapshot of the stream, let r be the region returned by Algorithm 5, and r_{opt} be the bursty region returned by our exact solution. We have $\mathcal{S}(r) \geq \frac{1-\alpha}{4}\mathcal{S}(r_{opt})$. $\square$

Proof:

Since the sizes of r_{opt} and any cell are both $a \times b$, then r_{opt} either overlaps with a cell or intersects with four cells. We consider the following two cases.

Case 1: r_{opt} overlaps one cell. Since we return the candidate with maximum burst score, we will return the bursty region to users. The approximate ratio is 1.

Case 2: r_{opt} intersects with 4 cells. Consider the example shown in Figure 4.4. Let the solid line rectangle be r_{opt} . The four dashed line rectangles are four cells which intersect with r_{opt} . According to Lemma 18, we have $(1 - \alpha)\mathcal{S}(r_{opt}) \leq \mathcal{S}(c_1 \cup \dots \cup c_4)$. According to Lemma 19, we can derive that $\mathcal{S}(c_1 \cup \dots \cup c_4) \leq \sum_{i \in [1, 4]} \mathcal{S}(c_i)$. Since we report the cell with the maximum burst score, i.e., $\mathcal{S}(r) \geq \mathcal{S}(c_i)$ for any $i \in [1, 4]$. Thus, we have $\frac{1-\alpha}{4}\mathcal{S}(r_{opt}) \leq \mathcal{S}(r)$.

Lemma 20: The approximation ratio is tight. $\square$

Proof: We show the approximation ratio is tight by giving an example. Consider an instance in Figure 4.5, where c_1, c_2, c_3 and c_4 are cells in the grid, and the solid-line

Figure 4.4: Proof for Theorem 10

Figure 4.5: A tight example.

rectangle r_{opt} is the bursty region with the maximum burst score. The white nodes are the spatial objects in window W_c and the black nodes are in W_p . We assume that $\frac{o.w}{|W_c|} = \frac{o.w}{|W_p|} = 1$ for each object o . The burst score for the region r_{opt} is $\mathcal{S}(r_{opt}) = \alpha \max(4 - 0, 0) + (1 - \alpha)4 = 4$. The burst score of cell c_i is $\mathcal{S}(c_i) = \alpha \max(1 - 1, 0) + (1 - \alpha) = 1 - \alpha$, for any $i \in [1, 4]$. Thus, the approximation ratio is tight.

Time Complexity. In Algorithm 5, it takes constant time to locate the cell and update the burst score. In our implementation, we use a heap to maintain all cells according to their burst scores. Let n be the number of spatial objects created in W_c and W_p . Since there are $O(n)$ non-empty cells, it takes $O(\log n)$ time to report the cell with the maximum burst score.

4.3.2 A Multi-Grid-Based Solution

The burst score of the region returned by Algorithm 5 is highly dependent on the position of the grid. In this subsection, we adopt multiple grids to further improve the result quality.

In the grid-based solution, we use a grid defined by lines

$$\text{Grid 1: } x = i \cdot b, y = i \cdot a$$

for all integers $i \in [-\infty, +\infty]$. By shifting the grid, we generate three additional grids for all integers $i \in [-\infty, +\infty]$:

$$\text{Grid 2: } x = 0.5b + i \cdot b, y = i \cdot a,$$

$$\text{Grid 3: } x = b + i \cdot b, y = 0.5a + i \cdot a,$$

$$\text{Grid 4: } x = 0.5b + i \cdot b, y = 0.5a + i \cdot a,$$

The multi-grid-based solution (called the mGAP-SURGE algorithm) invokes Algorithm 5 four times by using the four different grids. Among the four returned regions,

Algorithm 6: MGAP-SURGE Algorithm

Input: Spatial Object o **Output:** A region of size $a \times b$

```
1  $r_1 \leftarrow \text{GAP-SURGE}(\text{Grid1}, o, H_1);$   
2  $r_2 \leftarrow \text{GAP-SURGE}(\text{Grid2}, o, H_2);$   
3  $r_3 \leftarrow \text{GAP-SURGE}(\text{Grid3}, o, H_3);$   
4  $r_4 \leftarrow \text{GAP-SURGE}(\text{Grid4}, o, H_4);$   
5 return The region among  $r_1 \dots, r_4$  with highest burst score.
```

the one with the maximum burst score is returned to users. The pseudocode of the MGAP-SURGE algorithm is reported in Algorithm 6.

Theorem 11: The approximate ratio of the MGAP-SURGE algorithm is $\frac{1-\alpha}{4}$. $\square$

Proof: Since the MGAP-SURGE returns the best result of found by Algorithm 5, its approximation ratio is $\frac{1-\alpha}{4}$.

Time Complexity. MGAP-SURGE invokes Algorithm 5 four times, and its complexity is $O(\log n)$, where n is the number of spatial objects created in W_c and W_p .

4.4 Top-k Bursty Region Detection

Recall that in Example 3 in Chapter 1, it is paramount to monitor regions with outbreak of diseases. Intuitively, monitoring only the most bursty region is not sufficient. In fact, it is reasonable to be interested in a small list of such bursty regions. Specifically, given the size of a region, we need to continuously monitor the top- k regions of the given size with highest burst scores. In this section, we present how we can elegantly extend our proposed solutions to continuously detect top- k regions with highest burst scores. We begin by formally defining the *top-k bursty regions*.

4.4.1 Definition

Although at first glance it may seem that it is easy to define *top-k bursty regions*, in reality it is tricky. First of all, are the top- k regions allowed to overlap? It may seem that detecting k non-overlapping regions is a good choice. However, the non-overlapping requirement may lead us to overlooking some highly bursty regions. Hence, it is beneficial to allow the top- k bursty regions to be overlapping instead of disjoint in nature.

Next, how do we define the burst scores for two overlapped regions? For example, if a spatial object lies at the intersection of two overlapping regions, which region's burst score should it contribute to? A naïve idea is to consider it in both regions. However,

this may result in k regions that are highly similar to one another. To resolve this issue, we ensure that a spatial object contributes only to the burst score of at most one region.

The aforementioned considerations lead us to a greedy strategy for defining the *top- k bursty regions*. Specifically, given the first i bursty regions, the $(i + 1)$ -th bursty region is the region with maximum burst score in the space but excluding all spatial objects that are already covered by the first i bursty regions.

Definition 17:Top- k Bursty Regions. Given k rectangular regions $r_1, \dots, r_k$ such that each has a size of $a \times b$, we say $r_1, \dots, r_k$ are the top- k bursty regions if and only if for any region r of size $a \times b$, we have $\mathcal{S}(r_i \setminus r_{[1,i-1]}) \geq \mathcal{S}(r \setminus r_{[1,i-1]})$ for $i \in [1, k]$, where $r_{[1,i-1]}$ is union of regions $r_1, \dots, r_{i-1}$. $\square$

In order to address the top- k bursty regions problem, we reduce the top- k bursty regions problem to k CSPOT problems following the reduction in Section 4.2.1. The $(i + 1)$ -th CSPOT problem aims to detect the $(i + 1)$ -th bursty point from the space that excludes the set of rectangles that cover the top- i bursty points.

Observe that Definition 17 essentially paves the way to a greedy approach for selecting top- k bursty regions. Whenever an event happens, we can first detect a region with the maximum burst score by invoking Algorithm 3. Then we remove the spatial objects covered by the region. After that, we detect a region with the maximum burst score over the remaining objects. We repeat this process until k regions are selected.

However, the naïve strategy is inefficient as there are too many redundant computations, i.e., it is possible that we search a cell in all the k reduced CSPOT problems. To address the k CSPOT problems efficiently, we want to share the common computations among the k CSPOT problems.

4.4.2 Extension of the Exact Solution

In the extension of our exact solution, for each cell c , we maintain k upper bounds and k candidate points in order to solve the k CSPOT problems by following the idea of Algorithm 4. For each CSPOT problem, we adopt the lazy update strategy to access the cells in descending order of their upper bounds. If the candidate point of the top cell is not valid, we search the cell by invoking Algorithm 3.

We develop two ideas of sharing computation among the k CSPOT problems. Firstly, if a rectangle object can cover the i -th bursty point, it will not be considered in the CSPOT problems with order higher than i . For the extension, we maintain a level, denoted by $g.lvl$, for each rectangle object g . To select the i -th bursty point in response to a new event, we consider the set of rectangles $G[i : k]$ whose levels are no smaller than i , i.e., $G[i : k] = \{g | g.lvl \geq i\}$. When the i -th bursty point is selected, the levels of all the rectangles that cover the i -th bursty point are set as i , and these rectangles will not be

Algorithm 7: CCS-KSURGE Algorithm

Input: An event $e = \langle g, l \rangle$
Output: A bursty point

```

1  $g.lvl = k, V = \{g\};$ 
2 for  $i \in [1, k]$  do
3    $p_{old} = p[i];$ 
4    $C \leftarrow$  cells that are overlapped with  $V$ ;
5   for  $c \in C$  do
6     | Update  $U(c)[j]$  and  $c.p[j]$  for  $j \in [i, k]$ ;
7    $c \leftarrow \arg \max U(c)[i];$ 
8   while  $c.p[i]$  is invalid do
9     |  $c.p[i] \leftarrow \text{SL-CSPOT}(c)$  over  $G[i :];$ 
10    |  $U_d(c)[i] = \mathcal{S}(c.p[i])$  over  $G[i :];$ 
11    | if no rectangle in  $c$  covers any of  $p[1 : k]$  then
12      | |  $c.p[1 : k] = c.p[i], U_d(c)[1 : k] = U_d(c)[i];$ 
13    |  $c \leftarrow \arg \max U(c)[i];$ 
14    $p[i] \leftarrow c.p;$ 
15   Mark  $o.lvl = k$  for any  $o \in G(p_{old})[i] \setminus G(p[i])[i : k];$ 
16   Mark  $o.lvl = i$  for any  $o \in G(p[i])[i : k];$ 
17    $V \leftarrow G(p[i])[i : k] \cup G(p_{old})[i];$ 
18 return  $p[1 : k]$ 

```

considered by the CSPOT problems with a higher order than i . Meanwhile, if a rectangle covers the old i -th bursty point, but not the new i -th point, its level is reset to k so that it will be considered in all the k CSPOT problems.

Secondly, if no rectangle in a cell covers any of the k detected bursty points, all the rectangles in the cell will be considered in all k CSPOT problems. Thus, the upper bounds and the candidate points w.r.t. the k CSPOT problems for the cell are the same. That is, once the upper bound and the candidate point for the cell are computed for one CSPOT problem, we do not need to recompute them again for other CSPOT problems.

Algorithm 7 presents the detail of our extension. It takes as input an event $e = \langle g, l \rangle$, and output the top- k bursty points, denoted by $p[1 : k]$. It uses V to denote the set of objects that need to be handled subsequently, and is initialized as $\{g\}$ (line 1). It then solves the k CSPOT problem iteratively (lines 2–17). In each CSPOT problem, it first locates the set of cells affected by the objects in V (line 4). For each cell $c \in C$, the upper bound $U(c)[j]$ and candidate point $c.p[j]$ w.r.t. the j -th CSPOT problem are updated for $j \in [i, k]$ (lines 5–6). Then it accesses the cells in descending order of their upper bounds w.r.t. the i -th CSPOT problem (lines 8–14). The upper bound and candidate point are updated as in Algorithm 4 (lines 9–10). If no rectangle in cell c covers any of the k detected bursty points, its k upper bounds and candidate points are set to the same (lines 11–12). When a new bursty point is found, we reset the levels

for the affected objects as discussed earlier (lines 15–16): The rectangles that cover the old bursty point p_{old} but not the new bursty point $p[i]$ are newly visible to all the k cSPOT problems, while the rectangles that cover the new bursty point $p[i]$ are newly invisible to the cSPOT problems with a higher order than i . The two types of rectangle objects comprise V , which will be processed in the next cSPOT problem (line 17). After k iterations, it returns the top- k bursty points $p[1 : k]$ as the result.

Time Complexity. A cell is searched if its upper bound is either changed by an event or by a detected bursty point. Thus, the algorithm searches $O(1 + k) = O(k)$ cells on average when processing a rectangle. The complexity of Algorithm 7 is $O(|c_{max}|^2 \cdot k)$, where $|c_{max}|$ is the maximum number of objects that we search in a cell.

4.4.3 Extension of the Approximate Solutions

We also extend our approximate solutions in Section 4.3 to find k regions with relatively high burst score.

Extending the gap-surge Algorithm. Consider the grid-based solution. We use a heap to maintain all cells with their burst scores. Thus, we can simply return top- k cells with highest burst scores. In our implementation, we use a heap to maintain the cells. The algorithm is outlined in Algorithm 8. Thus, its complexity is $O(\log n)$.

Algorithm 8: GAP-KSURGE Algorithm

Input: Spatial Object o , Heap H , integer k , *Grid*

Output: A region of size $a \times b$

```

1  $c_{i,j} \leftarrow$  the cell  $o$  lies in;
2 if  $c_{i,j}$  not in  $H$  then
3   | add  $c_{i,j}$  to  $H$ ;
4 if  $o.t_c \in W_c$  then
5   |  $r_{can}.S_c + = \frac{o.\rho}{|W_c|}$ ;
6 else if  $o.t_c \in W_p$  then
7   |  $r_{can}.S_c - = \frac{o.\rho}{|W_c|}, r_{can}.S_p + = \frac{o.\rho}{|W_p|}$ ;
8 else
9   |  $r_{can}.S_p - = \frac{o.\rho}{|W_p|}$ ;
10  $r_{can}.S = \max(r_{can}.S_c - r_{can}.S_p, 0) + r_{can}.S_c$ ;
11 update  $H$ ;
12 return Top- $k$  cells in  $H$ 
    
```

Extending the mGap-surge Algorithm. We extend the multi-grid-based solution similarly. Note that one cell in a grid may overlap with at most four cells in another grid. Thus, for each grid, we maintain the top- $4k$ cells. Then we merge the $16 \cdot k$ cells and return the top- k non-overlapping cells. The algorithm is outlined in Algorithm 9. Its time complexity is $O(\log n + k)$.

Algorithm 9: MGAP-KSURGE Algorithm**Input:** Spatial Object o **Output:** A region of size $a \times b$

- 1 $l_1 \leftarrow \text{GAP-SURGE}(\text{Grid1}, o, H_1, 4k);$
- 2 $l_2 \leftarrow \text{GAP-SURGE}(\text{Grid2}, o, H_2, 4k);$
- 3 $l_3 \leftarrow \text{GAP-SURGE}(\text{Grid3}, o, H_3, 4k);$
- 4 $l_4 \leftarrow \text{GAP-SURGE}(\text{Grid4}, o, H_4, 4k);$
- 5 $r[1, k] \leftarrow \text{top-}k \text{ non-overlapping cells from } l_1 \cup l_2 \cup l_3 \cup l_4;$
- 6 **return** $r[1, k]$

Table 4.2: Datasets.

Datasets	UK	US	Taxi
# of Spatial Objects	1,000,000	1,000,000	1,000,000
Arrival Rate(per hour)	5,747	16,802	18,145
Range of Latitude	139.0 150.9	100.1 150.4	41.6 42.2
Range of Longitude	171.1 181.9	40.2 118.8	12.0 12.9

4.5 Experimental Study

We investigate the performance of our proposed techniques. All algorithms are implemented in C++ complied with GCC 4.8.2. The experiments are conducted on a machine with a 2.70GHz CPU and 64GB of memory running Ubuntu.

4.5.1 Experimental Setup

Datasets. We conduct experiments on three public real-life datasets as reported in Table 4.2. UK consists of 1,000,000 geo-tagged tweets posted in UK. US consists of 1,000,000 geo-tagged tweets posted in US and has a higher arrival rate. Taxi¹ consists of mobility traces of taxi cabs obtained from the GPS in Roma, Italy. It contains 1,000,000 records over 5 days. For each dataset, the weight of each spatial object is randomly chosen from from $[1, 100]$ with a uniform distribution.

Algorithms. We evaluate the performances of the three proposed algorithms, namely the exact method *Cell-CSPOT* (denoted by CCS), the grid-based approximation algorithm GAP-SURGE (denoted by GAPS), and the multi-grid-based technique MGAP-SURGE (denoted by MGAPS). We denote the top- k extensions of these algorithms as kCCS, kGAPS, and kMGAPS, respectively. To evaluate the usefulness of our proposed method of upper

¹crawdad.org/roma/taxi/20140717

bound estimation, we compare **CCS** with an approach that only utilizes the static upper bound. We denote this baseline method by **B-CCS**. We also compare **CCS** with a baseline approach that does not use any upper bound estimation technique, denoted by **Base**. Specifically, in **Base** we divide the space into cells, and we search all the cells that overlap with the rectangle object when an event happens. To the best of our knowledge, there is no existing technique that addresses the **SURGE** problem. Hence we are confined to compare our proposed algorithms with **aG2** [AH16], which is designed for continuously monitoring the **MaxRS** problem. Obviously, we cannot directly apply it to solve the **SURGE** problem. In our experiments, we use a modified version of **aG2**. Specifically, the modified algorithm inherits the grid index structure and the branch-and-bound strategy from the original algorithm. The main difference between the modified and the original algorithms is how we search a rectangle object given a snapshot of the stream. In the original algorithm, they invoke the sweep-line algorithm [NB95] to search a rectangle object to find a region with maximum sum score, while in the modified algorithm, we use our proposed **SL-CSPOT** algorithm instead. With a slight abuse of notation, we still use **aG2** to denote the modified **aG2**.

Parameters. By default, we set the size of the past window W_p and the current window W_c as 1 hour for **US** and **UK**, and 5 minutes for **Taxi**. We set the size of the query rectangle as $1/1000$ of the range of each dataset by default, denoted by q . We set the preferred area A as the whole space. For the **aG2** algorithm, we set the size of a cell to $10q$.

Stream Workload. We start the simulation when the system becomes stable, i.e., there exists an expired object from the past sliding window. We continuously run each algorithm for 1,000,000 new arriving spatial objects over the two sliding windows. The average processing time per object is reported.

4.5.2 Evaluation of the Exact Solution

We first evaluate the runtime performance of **CCS**, **B-CCS** and **Base** on each dataset. Then we study the usefulness of the upper bound in **CCS**.

Runtime Performance. The aim of the first set of experiments is to evaluate the efficiency of our exact solution *w.r.t.* the sliding window size and the query rectangle size. For **US** and **UK**, we vary the sliding window with the following sizes: *30 minutes*, *1 hour*, *2 hours*, *5 hours*, and *12 hours*. For **Taxi**, we use the following five sizes for sliding windows: *1 minute*, *5 minutes*, *10 minutes*, *20 minutes*, and *30 minutes*. We use the following four sizes for the query rectangle: $0.5q$, q , $2q$, and $3q$.

Figures 4.6(a)–(c) report the average runtime of the three methods for processing one spatial object as we vary the size of sliding windows. Note that the y-axis is in logarithmic scale. We find that **CCS** runs efficiently and outperforms **aG2**. For example,

Figure 4.6: Runtime of CCS, B-CCS, Base and aG2.

for **Taxi**, it takes about 3×10^{-4} seconds to process an object when the current and past windows are both set to 30 minutes, while **aG2** takes 7×10^{-3} seconds. In addition, we find that **aG2** runs out of the 64 GB memory on **US** when the current window and past window are both set as 12 hours, as there are too many spatial objects in the two sliding windows.

Moreover, we observe that the processing time per object of all algorithms increases as the size of window increases. This is due to the need to consider a larger number of spatial objects when we search for the bursty region with the increase in size of the sliding window. Consequently, the runtime per object increases.

Figures 4.6(d)–(f) report the average runtime for processing one spatial object as we vary the size of the query rectangle. Similarly, the average runtime increases as size of the rectangle increases. We observe that the **aG2** outperforms the baseline algorithm for datasets **Taxi** and **UK**, while it is worse than baseline for **US**. This is because there are 3,024 and 11,494 spatial objects in the current and the past windows on average for **Taxi** and **UK**, respectively, while there are 33,604 spatial objects for **US**. Recall that **aG2** needs to maintain the overlapping graph for such spatial objects. The runtime of **aG2** degrades faster than **base** as the average number of the spatial objects in the sliding windows increases.

Table 4.3: Ratio of rectangle messages that trigger a search vs. window size for CCS and B-CCS.

Taxi	Window (mins)	1	5	10	20	30
	CCS	4.85%	3.20%	2.56%	2.13%	1.95%
	B-CCS	92.63%	78.30%	70.00%	62.07%	57.90%
UK	Window (hours)	0.5	1	2	5	12
	CCS	0.34%	0.27%	0.23%	0.37%	0.48%
	B-CCS	37.79%	28.23%	22.76%	21.64%	14.57%
US	Window (hours)	0.5	1	2	5	12
	CCS	0.60%	0.68%	0.70%	0.52%	0.60%
	B-CCS	64.21%	52.29%	35.13%	9.0%	20.90%

Usefulness of Upper Bound. Next, we evaluate the usefulness of the method for upper bound estimation in CCS. In this set of experiments, we process 1,000,000 new objects and report how many rectangles trigger a search. The results are reported in Table 4.3. Clearly, only a small portion of rectangle messages (2%–5% for Taxi, and less than 1% for US and UK) trigger a search in CCS compared with B-CCS. This is because CCS can estimate a much tighter upper bound for cells. Thus, many cells are eliminated from further checking. This also explains why CCS is much more efficient than B-CCS. As shown in Figure 4.6, we observe that CCS is more efficient than the other two methods. The runtime of CCS is more than one order of magnitude faster than B-CCS and Base, respectively. Moreover, we observe that B-CCS is only marginally better than BASE, which indicate that only using the static upper bound cannot effectively avoid unnecessary recomputation. This is because the static upper bound is too loose, especially when the weights of the objects are randomly chosen from 1 to 100.

4.5.3 Evaluation of the Approximate Solutions

Approximate Ratio. In this set of experiments, we vary the sliding window to assess the approximate ratio of the burst scores of region detected by GAPS and MGAPS. The detailed results are reported in Table 4.4. Though the theoretical approximate ratio is $\frac{1-\alpha}{4}$, in practice it is much better, especially for MGAPS. We observe that for UK, the burst score of the region detected by GAPS is about 70%–90% of the burst score of the optimal region. The region detected by MGAPS is about 85%–95% of the burst score of the optimal region. Since GAPS and MGAPS are much more efficient than CCS (about three orders of magnitude faster), they are good alternatives to CCS when a slight imprecision is acceptable.

Runtime Performance. We evaluate the efficiency of our approximate solutions *w.r.t.* the sliding window size and the query rectangle size under the same setting as for the exact solution. Figures 4.7 (a)–(c) report the average runtime for processing one spatial object using GAPS and MGAPS as we vary the sliding window. Figures 4.7 (d)–(f) report

Figure 4.7: Runtime performance of GAPS and MGAPS.

the average runtime for processing one spatial object as we vary the size of the query rectangle. We find that the runtime of MGAPS is about 2-5 times of GAPS, which is expected as MGAPS invokes GAPS four times. Moreover, GAPS and MGAPS are about three orders of magnitude faster than CCS by comparing Figure 4.6 and Figure 4.7.

4.5.4 Effect of α

In the definition of burst score, we use a parameter α to balance the significance and the burstiness. In this set of experiments, we evaluate the impact of the parameter α on the efficiency and approximation ratio of our proposed algorithms on the US dataset. We use 1 hour for the sliding windows and q for the size of the query rectangle.

Impact on Runtime Performance. We evaluate the efficiency of our exact and approximate solutions w.r.t. the balance parameter α . Figure 4.8 reports the average runtime for processing one spatial object as we vary α from 0.1 to 0.9. We observe that the efficiency is hardly affected by the parameter α for both our exact solution and approximate solutions.

Impact on Approximation Ratio. In this set of experiments, we evaluate the approximate ratio of the burst scores of regions detected by GAPS and MGAPS by varying α . The results are reported in Table 4.5. We find that the approximate ratios of the two

Table 4.4: Approximate ratio vs. the size of window.

Taxi	Window (mins)	1	5	10	20	30
	GAPS	76.34%	73.90%	75.12%	75.70%	76.35%
	MGAPS	85.98%	85.14%	87.35%	88.34%	87.85%
UK	Window (hours)	0.5	1	2	12	24
	GAPS	90.22%	91.56%	91.98%	89.82%	92.44%
	MGAPS	93.13%	94.34%	93.76%	90.50%	92.82%
US	Window (hours)	0.5	1	2	12	24
	GAPS	84.23%	80.67%	89.70%	91.77%	80.10%
	MGAPS	88.61%	88.07%	91.44%	91.77%	84.34%

Figure 4.8: Runtime performance w.r.t. α on US.

algorithms decrease as α increases. This is because their theoretical approximate ratio $\frac{1-\alpha}{4}$ decreases as α increases.

4.5.5 Scalability

We now investigate the scalability of our proposed techniques by varying the arrival rate of the spatial objects. Specifically, we use 1 hour for both the current window and past window, and q for the size of the query rectangle on all three datasets. We stretch the stream to change its arrival rate from 2 million per day to 10 million per day. For example, in UK, 1 million spatial objects arrived in 174 hours. Hence, we shrink the arrival time of each object to make all objects arrive in 24 hours. Then the arrival rate of the stream is 1 millions per day. We only report the *average time* for processing the objects arrived in one hour (denoted by t_h) of CCS and GAPS in Figure 4.9. Formally, $t_h = \frac{runtime}{|\mathcal{O}|_{time}}$, where *runtime* is the runtime of the algorithm, and $|\mathcal{O}|_{time}$ is the total timespan of the stream.

We observe that it takes several hours for CCS to process the objects arrived in an hour for the Taxi dataset, which means that it does not scale well and cannot handle

Table 4.5: Approximate ratio vs. α .

	α	0.1	0.3	0.5	0.7	0.9
US	GAPS	82.57%	81.76%	80.67%	77.23%	78.58%
	MGAPS	90.50%	89.44% ^b	88.07%	87.80%	86.67%

Figure 4.9: Scalability study.

streams with high arriving rate. On the other hand, our approximate solutions, GAPS and MGAPS, scale well with the increase in arrival rate. They can process the objects arrived in an hour within seconds.

4.5.6 Finding Top- k Bursty Regions

We next evaluate the performance of the extensions of our three algorithms for continuously detecting top- k bursty regions. We study the effect of k and the size of sliding windows.

Runtime Performance. This set of experiments aims to evaluate the efficiency of these algorithms *w.r.t.* the sliding window size. We adopt the same setting as in Section 4.5.2. Figures 4.10(a)–(c) report the average runtime per object of kCCS, kGAPS, and kMGAPS for different sliding windows. We observe that as the sliding window gets larger, kCCS does not scale well and cannot process the top- k queries efficiently. Meanwhile, kGAPS and kMGAPS can find top- k bursty regions efficiently.

We also compare the naïve solution for finding top- k bursty regions with these algorithms. Recall from Section 4.4, in the naïve solution, we detect the top- k bursty regions for each newly-arrived object. Clearly, the naïve solution is prohibitively expensive. Hence, we only run it with a small sliding window on US, and its runtime per object is about 100X more than kCCS.

 Figure 4.10: Top- k bursty regions detection.

Effect of k . Next, we study how the value of k affect the runtime performance of the three extensions. We use the following 4 values for k : 3, 5, 7 and 9. The runtime performance is depicted in Figures 4.10(d)–(f). We observe the runtime per object of kCCS increases as k increases. This is because we divide the top- k bursty region detection problem into k instances of bursty region detection problems. Each bursty region detection problem takes $O(n_c^2)$ time to find a bursty region, where n_c is the number of spatial objects in the cells that we actually searched. In addition, we also observe that kGAPS and kMGAPS are less affected by k .

4.5.7 Case Study

To evaluate the result quality of our *cell*-CSPOT algorithm, we conduct a case study on the region monitored by the algorithm. We run the *cell*-CSPOT algorithm on the tweets posted in United States from 2012 April to 2012 October. We use the default parameter q as the size of the query rectangle. We set 24 hours as the size of the current and the past windows. The balance parameter α is set as 0.5. Note that since the algorithm continuously reports the location of bursty regions, we only present two examples of the detected bursty region and explain the connection between the region and real life events.

Example 20: In the first example, we present detecting bursty regions about “concert”. Specifically, we only consider tweets containing keyword “concert” and continuously report the detected bursty region. On July 8, 2012, our algorithm detected a region as

shown in Figure 4.11. The frequent keywords in this region during this time are “Walt” and “Concert”. By checking the events that happened in July 2012, we find that there was a concert performed by Ketherine Eason with Inner City Youth Orchestra of Los Angeles in Walt Disney Concert Hall in the detected region. $\square$

Example 21: In the second example, we present detecting bursty regions about “parade”. On May 19, 2012, our algorithm detected a region as shown in Figure 4.12. The frequent keywords in this region are “annual”, “dance”, and “parade”. By checking the events that happened in May 2012, we noticed that the dance parade is an annual parade and festival in New York. Specifically, in 2012 the parade took over Broadway Street on May 19th. $\square$

4.6 Summary

The work reported in this chapter is motivated by new opportunities brought by the massive volumes of streaming geo-tagged data (i.e., spatial objects) generated by location-enabled mobile devices. Specifically, we have studied a new problem called the SURGE problem to continuously detect the bursty region in a given area in real time. The SURGE problem is important as it can underpin various applications such as disease outbreak detection. We have proposed an exact solution and two approximate solutions for SURGE. We have also extended these solutions to find top- k bursty regions. Finally, our experiment study with real-world datasets has demonstrated the efficiency of our framework. As part of future work, we intend to explore the SURGE problem in the context of road network.

Figure 4.11: Bursty Region about “concert”.

Figure 4.12: Bursty Region about “parade”.

Chapter 5

Attribute-based Similar Region Detection

This chapter defines and studies a novel problem called the attribute-based similar region search (ASRS) problem. Specifically, given a query region of size $a \times b$ and an attribute A , the ASRS problem aims to identify a region of size $a \times b$ from a preferred area such that the data distribution of the attribute A in the region is most similar to the query region. To address the ASRS problem, we propose a grid-based solution. Specifically, we first impose a grid over the space. For each cell, we estimate a lower bound for the attribute distance. The cells are searched by a sweep line based algorithm in a greedy manner. We conduct experiments to show that our proposed algorithm runs efficiently.

This chapter is organized as follows. We formally introduce the ASRS problem in Section 5.1. We then present the exact solution to the ASRS problem in Section 5.2. The experimental study is presented in Section 5.3.

5.1 Problem Statement

In this section, we formally formulate our attribute-based similar region search problem. Firstly, we define some terminology that will be used to define the ASRS problem.

5.1.1 Terminology

Definition 18:Attributes Set Let A_i be an attribute, and $dom(A_i)$ be the domain of the attribute A_i . An attribute set $\mathcal{A} = \{(A_1, V_1), \dots, (A_d, V_d)\}$ is a set of key-value pairs, where A_i is an attribute and V_i is the set of values of attribute A_i , *i.e.*, $V_i \subseteq dom(A_i)$. Given an attribute set $\mathcal{A}$, we use $\mathcal{A}[A_i]$ to denote the corresponding values V_i in $\mathcal{A}$, and we let $\mathcal{A}[A_i] = null$ if A_i is not a key in $\mathcal{A}$. $\square$

Table 5.1: Example of Spatial Objects.

	o_1	o_2	o_3	o_4	o_5
ρ	(1, 2)	(2, 3)	(2, 1)	(4, 3)	(4, 4)
$\mathcal{A}$	Categ.: Cafe Rating: 4 Price: 47	Categ.: Cafe Rating: 5 Price: 32	Categ.: Theater Rating: 3 #Hall: 5	Categ.: Shop Rating: 5 Price: 70 #Visit: 500	Categ.: Shop Rating: 4 #Unit: 170 #Visit: 900

Definition 19: Spatial Object. A spatial object is represented with a tuple $o = \langle \rho, \mathcal{A} \rangle$, where ρ is the location point with latitude and longitude, and $\mathcal{A}$ is an attribute set which describes the properties of o . We say a spatial object o *has* an attribute A_i if $o.\mathcal{A}[A_i] \neq null$. $\square$

In real world, a spatial object can be associated with various attributes. For example, a point of interest (POI) can be viewed as a spatial object. It could have attributes like “category”, “rating”, and “average price”. Some of the attributes (like “category”) are categorical attributes, *i.e.*, the domain of the attribute is a finite set of values. Some other attributes (like “average price”) are continuous attributes, which has infinite values in its domain. In this work, we assume that every attribute is a categorical attribute. For the continuous attributes, we can discretize them to categorical attributes.

Note that in Definition 18, a spatial object is allowed to have a set of values for one attribute, which makes the definition of spatial objects more general. For example, we can view a geo-tagged tweet as a spatial object. The value of its attribute “words” is the set of words in the text of the tweet. Then given a query region, our ASRS problem can be used to identify a region with similar word distribution.

Example 22: Table 5.1 illustrates an example dataset consisting of five spatial objects $o_1, \dots, o_5$. In the dataset, object o_1 has three attributes: “Category”, “Rating”, and “Price”. Note that in the dataset, it is not required that all spatial objects have the same attributes. For example, object o_3 is a theater. Thus it has an attribute “#Hall”, which is the number of halls in the theater, and none of the other spatial objects has this attribute. $\square$

5.1.2 Data Distribution of an Attribute

Intuitively, the data distribution of an attribute reveals significant properties of the region *w.r.t.* the attribute. For example, in order to get an overview of the functionality of a region, people may want to know the data distribution of the *category* attribute. Meanwhile, people can evaluate the price level of a given region if they know the data distribution of the *average price* attribute. As a result, it is a fundamental task to get

Figure 5.1: Example.

the data distribution of a specified attribute. We define the distribution of an attribute as follows.

Definition 20:Attribute Distribution Consider a specified attribute A and a set of spatial objects O . The attribute distribution of O *w.r.t.* A , denoted by T , is a function $T : \text{dom}(A) \rightarrow I$ that maps a value $v \in \text{dom}(A)$ in the domain of A to the number of spatial objects in O that has v as a value for the attribute A , *i.e.*,

$$T(v) = |\{o \in O | v \in o.\mathcal{A}[A]\}|, \text{ for } v \in \text{dom}(A)$$

□

For a rectangular region r , let O_r be the set of spatial objects that are located in r . We can compute the attribute distribution for region r by on O_r . We use $r.T$ to denote the attribute distribution for region r .

Definition 21:Attribute Distance Consider a specified attribute A . For any two regions r_i and r_j , $i \neq j$, the distance between the attribute distributions of r_i and r_j is defined as

$$\text{dist}(r_i.T, r_j.T) = \sum_{v \in \text{dom}(A)} |r_i.T(v) - r_j.T(v)|$$

□

Example 23: Consider the example shown in Figure 5.1. The attribute sets of the five objects are given in Table 5.1. The attribute distribution *w.r.t.* *category* for region r_1 is $\{\text{Cafe} : 2\}$, while the attribute distribution for region r_2 is $\{\text{Theater} : 1, \text{Shop} : 2\}$. The domain of the attribute *category* is $\{\text{Cafe}, \text{Theater}, \text{Shop}\}$. Thus, the attribute distance between r_1 and r_2 is

$$\text{dist}(r_1.T, r_2.T) = \sum_{v \in \{\text{Cafe}, \text{Theater}, \text{Shop}\}} |r_1.T(v) - r_2.T(v)| = 2 + 1 + 2 = 5$$

□

Now we formally define the *attribute-based similar region search* (ASRS) problem:

Definition 22: The ASRS problem. Given a set of attributed spatial objects O , a query region r_q of size $a \times b$, a target attribute A , and a preferred area, the *attribute-based similar region search* (ASRS) problem is to find the location of a region r of size $a \times b$ in the preferred area, such that the attribute distance between the attribute distributions of r_q and r is minimized:

$$r = \arg \max dist(r_q.T, r.T)$$

□

5.2 A Proposed Solution

The ASRS problem requires us to find the region with the minimum attribute distance in the preferred area, which could be located in any point from the infinite points in the area. Obviously, it is prohibitively expensive to check the region located at each point. As a result, we present an exact solution to address the ASRS problem in this section. Specifically, we first introduce the *attribute-based rectangle intersection* (ARI) problem. We show that we can reduce the ASRS problem to the ARI problem. Then we propose a grid-based algorithm, which imposes a grid over the preferred area and computes a lower bound of attribute distance for each cell. We can greatly reduce the search space by pruning the cells whose lower bound is larger than the current result. Finally, we present a sweep-line based algorithm to search a cell to identify the region with minimum attribute distance.

5.2.1 The ari Problem

We next present the definition of the ARI problem and illustrate how to reduce the ASRS problem to the ARI problem.

Definition 23: The ARI problem. Consider a set G of rectangles of size $a \times b$, each of which is associated with an attribute set. We say a rectangle $g \in G$ covers a point p if p is located inside g . Let $G(p)$ be the set of rectangles that can cover the point p . The attribute distribution of p , denoted by $p.T$, is defined over $G(p)$. Given a point q in the space, the ARI problem aims to find a point p , such that the attribute distance between $p.T$ and $q.T$ is minimized:

$$p = \arg \max dist(q.T, p.T)$$

where $q.T$ and $p.T$ are the attribute distribution of q and p , respectively. □

We next illustrate the reduction from the ASRS problem to the ARI problem. For each spatial object o in the preferred area in the ASRS problem, we draw a rectangle g whose left-bottom corner is located at o . The generated rectangle g is associated with the

Figure 5.2: Reduce to ARI problem

attribute set of o , i.e., $g.\mathcal{A} = o.\mathcal{A}$. Consider the example in Figure 5.2. There are three objects o_1, o_2 and o_3 in the space. Each object has one attribute A_1 , and the value of the attribute is listed in the table. For object o_1 , we draw a rectangle g_1 of size $a \times b$ whose left-bottom corner is located at o_1 . We get g_2 and g_3 in a similar way. Each rectangle has the same attribute set as the corresponding object. Let the top-right corner of the query rectangle in the ASRS problem be the query point q in the ARI problem. Therefore we get an instance of the ARI problem.

We next show that the result of the reduced ARI problem is the top-right corner of the ASRS problem.

Theorem 12: Let p be the point with minimum attribute distance in the reduced ARI problem. The rectangle of size $a \times b$ whose top-right corner is located at p has the minimum attribute distance to the query region in the ASRS problem. $\square$

Proof: Let r be any rectangle of size $a \times b$ in the ASRS problem, and p be the top-right corner of r in the reduced ARI problem. For any spatial object o in the ASRS problem, it is inside r iff the corresponding rectangle g in the ARI problem can cover the point p . Since the spatial object o and the corresponding rectangle g have the same attribute set, the attribute distribution of region r and the point p are always the same. As a result, if the attribute distance between r and the query region r in the ASRS problem is minimized, then p has the minimum attribute distance to the query point q in the ARI problem as well.

Note that the reduction is inspired by the idea of reducing the max-enclosing rectangle problem to the *rectangle intersection problem* [NB95]. Given a set of rectangles with non-negative weights, the *rectangle intersection problem* aims to find the area such that the sum of the overlapped rectangles is maximized. In this problem, the sum function is monotonic. Specifically, given a set of objects, the sum of the set increases as we add new objects into the set. However, in our ARI problem, we use the attribute distance as the score function, which is not monotonic. Thus, the techniques designed for the

rectangle intersection problem cannot be applied in our ARI problem to search for the point with minimum attribute distance.

According to Theorem 12, we can address the ASRS problem by solving the ARI problem. We next justify the reason behind the reduction of the ASRS problem to the ARI problem. In the ARI problem, the edges of the rectangles divide the space into disjoint regions. Consider the example in Figure 5.2. The grey area is one of the disjoint regions which is the overlap of g_1 and g_2 . For any point p in the grey area, it is covered by g_1 and g_2 , i.e., $G(p) = \{g_1, g_2\}$. Hence, all points in the grey area have the same attribute distribution. Moreover, we have the following theorem to bound the number of disjoint regions.

Theorem 13: Given a set of rectangles, there are at most $O(n^2)$ disjoint regions, where n is the number of rectangle objects.[NB95] $\square$

According to Theorem 13, we only need to consider $O(n^2)$ disjoint regions in the ARI problem instead of the infinite points in the ASRS problem, which justifies the reduction of ASRS problem to the ARI problem.

Example 24: Consider an instance of the reduced ARI problem in Figure 5.2. Let the attribute distribution of the query point $q.T = \{v_1 : 2\}$. Let p_1 be a point in the grey area, which is covered by g_1 and g_2 . Thus, its attribute distribution is $p_1.T = \{v_1 : 2\}$. Point p_1 has the minimum attribute distance to the query point, and is the result of the reduced ARI problem. We draw a rectangle of size $a \times b$ whose top-right corner is located at p_1 (the solid line rectangle) and it is the result of the ASRS problem. $\square$

According to Theorem 13, an intuitive strategy to solve the ARI problem is as follows: Firstly, we find out all disjoint regions. Then for each disjoint region, we compute the attribute distribution for points inside the region. We next find the disjoint region with the minimum attribute distance, and return a point in the region as the final result for the ARI problem. However, it is very inefficient to examine every disjoint region since there are $O(n^2)$ regions in the worst case. In the rest of this section, We present a grid-based algorithm to address the ARI problem efficiently. Specifically, we present a method to estimate a lower bound of attribute distance for each cell (Section 5.2.2.2). Moreover, we introduce a sweep-line based algorithm to locate the point with the minimum attribute distance in the cell (Section 5.2.2.3).

5.2.2 A Grid-based Algorithm

In this section, we proceed to present a grid-based algorithm to address the ARI problem efficiently.

5.2.2.1 Overview

The key idea behind our grid-based algorithm for the ARI problem is as follows: By imposing a grid over the space, we divide the space into cells. We can estimate a lower bound of the attribute distance for each cell. Therefore, we can examine the cells in a greedy manner: the cells with smaller lower bounds are examined firstly. This idea raises two challenges: (1) How to estimate the lower bound of attribute distance for each cell? and (2) How can we examine a cell efficiently to locate the point with the minimum attribute distance?

To address the first challenge, i.e., estimating a lower bound of the attribute distance, we first need to estimate the attribute distribution of the cell. Specifically, for each cell c , we compute two bounds for the attribute distributions, denoted by $c.T_l$ and $c.T_u$. Let $p.T$ be the attribute distribution of any point p in the cell. For any value v in the domain, we have $c.T_l[v] \leq p.T[v] \leq c.T_u[v]$. We can leverage the two bounds to estimate the lower bound of the attribute distance. To address the second challenge, i.e., locating the point with the minimum attribute distance, we propose a sweep-line based algorithm. We next elaborate on these issues.

5.2.2.2 Estimate Lower Bound

We first introduce the definition of *grid*. Then we show how to compute the bounds $c.T_l$ and $c.T_u$ for the attribute distribution of any point p in the cell.

Definition 24: (α, β) -**grid**. A (α, β) -grid is define by lines

$$x = i \cdot \beta, y = i \cdot \alpha$$

for all integers $i \in [-\infty, \infty]$. The size of each cell in the grid is $\alpha \times \beta$. We refer to $\alpha \times \beta$ as the granularity of the grid. $\square$

We next introduce the *full-cover set* and *partial-cover set* for a cell.

Definition 25: **Full-cover set and partial-cover set**. Consider a cell c and the set G of rectangles in the ARI problem. We define its full-cover set $c.G_f$ as the set of rectangles that can fully cover c , i.e., $c.G_f = \{g \in G | \forall p \in c, p \in g\}$. We define its partial-cover set $c.G_p$ as the set of rectangles that can partial cover c , i.e., $c.G_p = \{g \in G | c \cap g \neq \emptyset \wedge \exists p \in c, p \notin g\}$ $\square$

With the definition of full-cover set and partial-cover set, we can compute two attribute distributions for a cell as follows.

Definition 26: **Lower and upper distribution**. Consider a cell c and a specified attribute A . The lower distribution of c , denoted by $c.T_l$, is the attribute distribution of c 's full-cover set *w.r.t.* attribute A , i.e.,

$$c.T_l(v) = |\{g \in c.G_f | v \in g.A[A]\}|, \text{ for } v \in \text{dom}(A)$$

The upper distribution of c , denoted by $c.T_u$, is the attribute distribution of the union of c 's full-cover set and partial-cover set *w.r.t.* attribute A , i.e.,

$$c.T_l(v) = |\{g \in c.G_f \cup c.G_p | v \in g.\mathcal{A}[A]\}|, \text{ for } v \in \text{dom}(A)$$

□

We can derive the following property for the lower and upper distribution of a cell c .

Lemma 21: Let p be a point in c , and $p.T$ is the attribute distribution of p *w.r.t.* the attribute A . We have

$$c.T_l(v) \leq p.T(v) \leq c.T_u(v)$$

□

Proof: Let $G(p)$ be the set of rectangles that can cover the point p . According to Definition 25, we have $c.G_f \subseteq G(p) \subseteq c.G_p \cup c.G_f$. Since $c.T_l$ and $c.T_u$ are the attribute distributions of $c.G_f$ and $c.G_f \cup c.G_p$, we have $c.T_l(v) \leq p.T(v) \leq c.T_u(v)$.

Lemma 21 shows that the lower and upper distribution of a cell c bounds the attribute distribution of any point p in the cell. Therefore, given the attribute distribution of the query point p and a cell c , we can derive the lower bound of attribute distance for any point p in the cell with the following lemma.

Lemma 22: Let $q.T$ be the attribute distribution of the query point q . Let p be any point in a cell c . For any value $v \in \text{dom}(A)$, we have

$$|q.T(v) - p.T(v)| \geq \begin{cases} c.T_l(v) - q.T(v), & \text{if } q.T(v) < c.T_l(v) \\ q.T(v) - c.T_u(v), & \text{if } q.T(v) > c.T_u(v) \\ 0, & \text{if } c.T_l(v) \leq q.T(v) \leq c.T_u(v) \end{cases} \quad (5.1)$$

□

According to Lemma 21, we can compute the lower bound of attribute distance for any point p in the cell by summing up $|q.T(v) - p.T(v)|$ for each value $v \in \text{dom}(A)$. With the lower bound of attribute distance, we can greedily examine the cells to find the point with the minimum attribute distance.

Granularity and Complexity. In order to estimate a tighter lower bound, the size of a cell in the grid should be smaller than the query rectangle so that there exist cells that can be fully covered by a rectangle. Usually, we can set the size of a cell as $\frac{a}{2} \times \frac{b}{2}$, where $a \times b$ is the size of the query rectangle. In this case, a rectangle can overlap with at most 9 cells in the grid. Thus, it takes $O(n)$ time to estimate the lower bounds for all cells.

Algorithm 10: SL-SEARCH Algorithm

Input: A cell c , a query point q
Output: A point p with minimum attribute distance

```

1  $p = null, d_{min} = \infty;$ 
2  $I_{init} \leftarrow$  initial interval on the sweep line;
3  $I_{init}.T \leftarrow$  attribute distribution over  $c.G_f$ ;
4 while sweep-line meets an horizontal edge of a rectangle  $r \in c.G_p$  do
5     Update the intervals on the sweep line ;
6     for each interval  $I$  on the sweep line do
7         Update the attribute distribution if  $I$  is covered by  $r$ ;
8          $d \leftarrow dist(I.T, q.T);$ 
9         if  $d < d_{min}$  then
10              $d_{min} \leftarrow d;$ 
11              $p \leftarrow$  a point on interval  $I$ ;
12 return  $p, d_{min};$ 
    
```

5.2.2.3 Search A Cell

We next address the second challenge, i.e., how to examine a cell to find a point with the minimum attribute distance.

The high level idea of the sweep-line algorithm is straightforward. We use a horizontal line, referred to as the sweep line, to scan the space top-down. The sweep line is divided into several intervals by the vertical edges of the rectangles that the sweep line crosses. For each interval, we maintain its attribute distribution and attribute distance to the query point. During the scanning, the point with the minimum attribute distance is recorded and returned.

Note that we only need to search inside the cell. Since all rectangles in $c.G_f$ can fully cover the cell, there is no need to consider the rectangles in $c.G_f$ during the searching. We can initialize the attribute distribution of the interval on the sweep line with the attribute distribution of $c.G_f$.

We next illustrate the algorithm with an example.

Example 25: Consider the example in Figure 5.3. Assume $c.G_f = \emptyset$, $c.G_p = \{g_1, g_2\}$. We consider the attribute A whose domain is $dom(A) = \{v_1, v_2\}$. The values of g_1 and g_2 's attribute A is shown in the Table. The attribute distribution of the query point q is $\{v_1 : 1, v_2 : 1\}$. We use a sweep line to scan the space top-down.

Consider the case as shown in Figure 5.3(a). Before the sweep line is going to meet the top edge of rectangle g_2 , the sweep line is divided into three intervals, $(-\infty, x_1]$, $(x_1, x_3]$, (x_3, ∞) , by the vertical edges of g_1 . Intervals $(-\infty, x_1]$ and (x_3, ∞) are not covered by any rectangle. Thus, their attribute distributions are both $\{v_1 : 0, v_2 : 0\}$. Interval $(x_1, x_3]$ is covered by g_1 , whose value of attribute A is v_1 . Thus, the attribute distribution of

Figure 5.3: Example of the Sweep line algorithm.

the interval is $\{v_1 : 1, v_2 : 0\}$. When the sweep line meets the top edge of rectangle r_2 , interval $(x_1, x_3]$ and interval (x_3, ∞) are divided into $(x_1, x_2]$, $(x_2, x_3]$ and $(x_3, x_4]$, (x_4, ∞) , respectively. The attribute distributions of $(x_1, x_2]$ (x_4, ∞) remain the same, while the intervals $(x_2, x_3]$ and $(x_3, x_4]$ are newly covered by g_2 . Thus, their attribute distributions are updated as $\{v_1 : 1, v_2 : 1\}$ and $\{v_1 : 0, v_2 : 1\}$, respectively. Interval $(x_2, x_3]$ has the minimum attribute distance seen so far. We record a point on this interval as the current result.

Then the sweep line meets the bottom edge of g_1 , as shown in Figure 5.3(b). After that, intervals $(-\infty, x_1]$ and $(x_1, x_2]$ are combined into one interval $(-\infty, x_2]$. Intervals $(x_2, x_3]$ and $(x_3, x_4]$ are combined into one interval $(x_2, x_4]$. We update their attribute distributions accordingly. We repeat this procedure until the whole space is scanned. The point with the minimum attribute distance is then returned. $\square$

Algorithm 10 outlines the pseudocode of the sweep line algorithm. It takes as input a cell c , and a query point q . It outputs a point p which has the minimum attribute distance to q in the cell.

Complexity There are $2 \cdot |c.G_p|$ horizontal edges. Whenever the sweep line meets a horizontal edge, it needs to scan all intervals and update the distribution for each interval. Thus it takes $O(|c.G_p|)$ time to process one horizontal edge. Therefore, the overall time complexity of Algorithm 10 is $O(|c.G_p|^2)$, where $|c.G_p|$ is the size of the partial-cover set of the cell.

5.2.2.4 The grid-based algorithm

We now have all components in place to present the GARI algorithm, which is outlined in Algorithm 11. It takes as input a set of rectangles G , a query point q , and an attribute A . It first builds the grid with a granularity of $\alpha \times \beta$ (line 2). Then for each rectangle r , it

Algorithm 11: GARI Algorithm**Input:** A set of rectangles G , a query point q , an attribute A **Output:** A point p with minimum attribute distance

```

1  $p_{min} = null, d_{min} = \infty;$ 
2  $C \leftarrow$  cells in a  $\alpha \times \beta$  grid;
3 for rectangle  $r \in G$  do
4    $C_p \leftarrow$  cells that are partially covered by  $r$ ;
5    $C_f \leftarrow$  cells that are fully covered by  $r$ ;
6   Update  $c.T_l$  for each cell  $c \in C_f$ ;
7   Update  $c.T_u$  for each cell  $c \in C_f \cup C_p$ ;
8 for cell  $c \in C$  do
9    $c.d \leftarrow$  lower bound of attribute distance by Lemma 22;
10 while  $\min_{c \in C}(c.d) < d_{min}$  do
11    $c \leftarrow \arg \min_{c \in C} c.d$ ;
12    $p, d \leftarrow \text{sl-Search}(c, q)$ ;
13   if  $d < d_{min}$  then
14      $d_{min} = d, p_{min} = p$ ;
15    $C = C \setminus \{c\}$ ;
16 return  $p$ ;
```

finds the cells that are fully covered and partially covered by r (lines 4–5). It then updates the lower distribution and upper distribution for the cells in C_f and $C_f \cup C_p$, respectively (lines 6–7). According to Lemma 22, it computes the lower bound of attribute distance for each cell (lines 8–9). After that, it examines the cells greedily (lines 10–15). Specifically, it first gets the cell with smallest lower bound (line 11). Then it invokes Algorithm 10 to find the point with the minimum attribute distance in the cell (line 12). Point p_{min} is updated if the returned point has a lower attribute distance (lines 13–14). The cell c is removed after the examination (line 15). This procedure is repeated until the minimum lower bound in C is larger than the current minimum attribute distance d_{min} . Point p_{min} is a result of the ARI problem.

Complexity It takes $O(n)$ time to compute the lower bounds of attribute distance for all cells. In each iteration, it takes $O(n_c^2)$ time to search a cell, where $|C_p|$ the number of rectangles in the partial-cover set. Thus, the overall time complexity of Algorithm 11 is $O(\max_{c \in C}(|c.G_p| \cdot n_c))$, where n_c is the number of cells that are examined, and $\max_{c \in C}(|c.G_p|)$ is the maximum size of the partial-cover set of all cells.

5.3 Experiments

In this section, we first present the setup of our experiments. Then we study the performance of our proposed algorithm *w.r.t.* the efficiency using real life datasets. All

Table 5.2: Dataset properties.

Property	Yelp	Gowalla
# of Objects	48,653	500,000
# of attributes	1	1
avg. # of values per attribute	3.4	1
$ dom(A) $	783	435

algorithms are implemented in C++ and compiled by VS 2015. All experiments are run on a Windows PC with Intel Xeon 3.70 GHz CPU and 64 GB memory.

5.3.1 Experimental Setup

Datasets We use two real life datasets in our experiments. Their properties are given in Table 5.2. YELP is a real-life dataset from the location-based service Yelp.com. Each spatial object is a POI. We consider one attribute “category” for the POIs. Each POI is associated with 3.4 categories on average. The domain of “category” has 783 different values. GOWALLA is a real-life dataset from a location-based social networking website www.gowalla.com. Each spatial object in the dataset is a POI. We consider one attribute “number of check-ins” for the POIs. Each object has one value for this attribute. The domain of the attribute has 435 different values.

Query Since we only have one attribute for all of our datasets, we only need to specify the size and the location of a query rectangle. Let $W \times H$ be the size of the minimum bounding rectangle of the dataset. We use $MBR/k = \frac{W}{k} \times \frac{H}{k}$ to denote the size of a query rectangle. For a given size of the query rectangle, we run a group of 10 queries, whose locations are randomly picked from the space.

Evaluated Algorithms To the best of our knowledge, there is no existing technique that address our ASRS problem. In our experimental study, we evaluate the following two algorithms. (a) Our proposed Algorithm 11, denoted by GARI, and (b) a baseline algorithm, denoted by BASE, which uses Algorithm 10 to search the whole space without the grid-based pruning technique.

5.3.2 Experimental Results

Efficiency In our first set of experiments, we evaluate the efficiency of our proposed algorithm *w.r.t.* the size of the query rectangle. We use the following 4 sizes for query rectangles, $MBR/500$, $MBR/1000$, $MBR/2000$, and $MBR/3000$. We set the granularity of the grid as $(w_q/2, h_q/2)$, where w_q and h_q is the width and height of the query rectangle, respectively. Figure 5.4 reports the runtime of the proposed algorithms for

Figure 5.4: Runtime vs. rectangle size.

query rectangles of different sizes. Note that it takes too much time for the BASE algorithm to run on the GOWALLA dataset. So we only report the runtime of the BASE algorithm on the YELP dataset. First, we observe that the runtime of GARI decreases as the size of the query rectangle increases. This is because when the size of the query is small, the granularity of the grid also small, which means there are more cells in the grid. As a result, more cells are likely to have small lower bounds of attribute distance. Hence, we have to search more cells when the size of the query rectangle is small. Second, we observe that the runtime of BASE increases as the size of the query rectangle increases. This is because there is no pruning technique in BASE and BASE has to scan the whole space. When the size of the query rectangle is large, two rectangles in the ARI problem are more likely to overlap with each other. Thus, it takes more time to process a horizontal edge of a rectangle, as it has a higher probability that the horizontal edge covers more intervals on the sweep line.

Effect of Grid Granularity In this set of experiments, we evaluate the effect of the grid's granularity. By using $MBR/1000$ as the size of the query rectangle, we use the following three granularities for the grid: $(w_q/2, h_q/2)$, $(w_q/3, h_q/3)$, and $(w_q/4, h_q/4)$, where w_q and h_q are the width and height of the query rectangle. Figure 5.5 reports the runtime of GARI *w.r.t.* the grid's granularity. We observe that as the granularity decreases, the runtime of GARI increases. This is because when the size of the cell gets smaller, there are more cells in the grid. It takes more time to estimate the lower bounds of attribute distance as a rectangle can affect more cells. In addition, it is likely that more cells will be searched since more cells will have smaller lower bounds of attribute distance.

Scalability Finally, we study the scalability of our proposed algorithm *w.r.t.* the number of spatial objects. Specifically, we generate three datasets by randomly drawing samples

Figure 5.5: Runtime vs. grid granularity.

from the GOWALLA dataset. On each dataset, we use $MBR/1000$ as the size of the query rectangle, and set the grid granularity as $(w_q/2, h_q/2)$, where w_q and h_q are the width and height of the query rectangle. We observe that the runtime of our proposed algorithm increases as the number of spatial objects increases. For the largest dataset with 700,000 POIs, GARI can be finished in reasonable time.

5.4 Summary

In this chapter, we introduce the attribute-based similar region search (ASRS) problem, which aims at finding a region with similar data distribution of a given attribute to the query region. To this end, we propose an exact algorithm called *GARI* which employs a grid-based strategy to prune unnecessary cells and a sweep line based algorithm to locate the point with the minimum attribute distance. We conduct experimental study to demonstrate the performance of the proposed algorithm. We show that the *GARI* algorithm can efficiently find the region with similar attribute distribution. We also show that the algorithm scales well *w.r.t.* the number of spatial objects.

Figure 5.6: Scalability.

Chapter 6

Influential Organizers Detection

This chapter proposes a novel problem to discover *influential event organizers* from online social networks who are essential to the overall success of social events. Informally, such event organizers comprise a small group of people who not only have the relevant skills or expertise that are required for an event (*e.g.*, conference) but they are also able to influence largest number of people to actively contribute to it. We formulate it as the problem of mining *influential cover set* (ICS) where we wish to find k users in a social network G that together have the required skills or expertise (modeled as attributes of nodes in G) to organize an event such that they can influence the greatest number of individuals to participate in the event. The problem is, however, NP-hard. Hence, we propose three algorithms to find approximate solutions to the problem. The first two algorithms are greedy; they run faster, but have no guarantees. The third algorithm is 2-approximate and guarantees to find a feasible solution if any. Our empirical study over several real-world networks demonstrates the superiority of our proposed solutions.

This chapter is organized as follows. Section 6.1 presents the independent cascade model, and the formal definition of the influential cover set problem. In Sections 6.2, we present two greedy algorithms, namely **ScoreGreedy** and **PigeonGreedy**. Section 6.3 presents the approximation algorithm **PICS**, its approximation ratio, and its optimized variant called **PICS+**. Section 6.4 presents the experimental study. Finally, we summarize the study in Section 6.5. The key notations used in this chapter are given in Table 6.1.

6.1 The Influential Cover Set (ICS) Problem

In this section, we begin by briefly introducing the *Independent Cascade* (IC) model [KKT03b] for influence propagation, which has been widely adopted in addressing the influence maximization problem (*e.g.*, [LKG⁺07, CWY09, CWW10b, KKY13, LCXZ12]). Specifically, we leverage this influence propagation model to define the influential cover set problem.

Notation	Definition
Q	The set of attributes in a query
k	The size of seed set
A	A set of attributes
$\mathcal{A}(v)$	The set of attributes associated to node v
$V(A)$	The set of nodes such that for any $v \in V(A)$, v can cover A
V_Q	The set of nodes such that for any $v \in V_Q$, $\mathcal{A}(v) \cap Q \neq \emptyset$
P	A partition
R	A covergroup
L_i	A list of tuples constructed with respect to $R.r_i$
t	A tuple in a list L
$\mathcal{T}$	A combination
$\sigma_{\mathcal{G}}(S)$	Influence spread of seed set S in $\mathcal{G}$
$\sigma_{\mathcal{G}}(S_1, S_0)^+$	$\sigma_{\mathcal{G}}(S_0 \cup S_1) - \sigma_{\mathcal{G}}(S_0)$, the influence increase of S_1 w.r.t. S_0

Table 6.1: Table of notations.

6.1.1 Independent Cascade (IC) Model

We refer to a social network as $\mathcal{G}(V, E, w)$, where V denotes a set of nodes and $E = \{(v_i, v_j) | v_i, v_j \in V\}$ is a set of edges. For each edge $(u, v) \in E$, $w(u, v)$ is the propagation probability from u to v , which ranges over $(0, 1]$. In the IC model, (a) each node is either *active* or *inactive*, and (b) a node is only allowed to switch from an inactive state to an active state, but not vice versa.

Given a seed set S and a social network $\mathcal{G}$, the IC model works in an inductive way: Let $S_0 = S$ and S_t denote the set of nodes activated at step t . At step $t + 1$, each node $v \in S_t$ has a single chance to activate each currently inactive neighbors u with a probability $w(u, v)$. The propagation process terminates when $S_t = \emptyset$.

The *influence spread* of S on $\mathcal{G}$, denoted as $\sigma_{\mathcal{G}}(S)$, is defined as the total number of nodes influenced by S . We define $\sigma_{\mathcal{G}}(S_1, S_0)^+ = \sigma_{\mathcal{G}}(S_1 \cup S_0) - \sigma_{\mathcal{G}}(S_0)$ to represent the marginal influence increase produced by S_1 w.r.t. S_0 . However, exact evaluation of influence spread under the IC model is #P-hard [CWW10b]. Monte-Carlo (MC) simulations method is proposed to obtain an accurate estimate of the influence spread [KKT03b]. By MC simulations method, we can obtain arbitrarily close approximation of influence with high probability.

In addition, heuristic approaches (*e.g.*, [LKG⁺07, CWY09, CWW10b, KKY13, LCXZ12]) have been proposed to approximate the computation.

6.1.2 Problem Definition

In practice, each node in a social network is usually associated with a set of attributes to describe the node, which can represent the skills or interests of a user. For instance, in

Meetup, users are associated with a set of interests like music and games. Given a social network $\mathcal{G}$ (or network for brevity), we denote the set of attributes of a node v by $\mathcal{A}(v)$. If attribute $a_i \in \mathcal{A}(v)$, we say that node v *has* (or *covers*) attribute a_i ; otherwise v does not have (or cover) attribute a_i . We say a set of nodes $V' \subseteq V$ cover a set of attributes $A = \{a_1, \dots, a_m\}$ if for any attribute $a_i \in A$, there exists at least one node $v \in V'$ such that v *has* a_i . We use $V(A)$ to denote the set of nodes such that each node in the set can cover A . Given a set of attributes Q , we define $V_Q \subseteq V$ the candidate set such that for any node $v \in V_Q$, v contains at least one attribute $a_i \in Q$. Hereafter, we refer to a network as $\mathcal{G}(V, E, w, \mathcal{A})$.

Example 26: Given a network containing 4 nodes v_1, v_2, v_3 and v_4 . $\mathcal{A}(v_1) = \{a, b, c\}$, $\mathcal{A}(v_2) = \{a, b\}$, $\mathcal{A}(v_3) = \{c, d\}$, and $\mathcal{A}(v_4) = \{a, b\}$. Let $A_1 = \{a\}$. Then $V(A_1) = \{v_1, v_2, v_4\}$. $\square$

It is natural to take the attributes into account in finding a set of most influential event organizers in a social network. As remarked earlier, the traditional influence maximization problem cannot be used to address this problem. Hence, in the following we propose the *influential cover set* problem to address it. Intuitively, the *influential cover set* is to find a set of k nodes in a social network such that the k nodes can cover the query attributes and the influence spread of the selected nodes is maximized.

Definition 27: Influential cover set (ICS). Given a set of attributes Q and parameter k , the influential cover set (ICS) problem aims to select the k seed nodes S from $\mathcal{G}(V, E, w, \mathcal{A})$:

$$\begin{aligned} S = & \arg \max_{S \subseteq V, |S|=k} \sigma_{\mathcal{G}}(S) \\ \text{s.t. } & Q \subseteq \bigcup_{s \in S} \mathcal{A}(s), \end{aligned} \quad (6.1)$$

where $\sigma_{\mathcal{G}}(S)$ is the influence spread of S on $\mathcal{G}$, *i.e.* the number of nodes influenced by nodes in S , and $\mathcal{A}(s)$ denotes the set of attributes associated with node s . $\square$

Note that $|Q|$ is typically small in our problem setting and hence we treat $|Q|$ as a constant. Furthermore, as mentioned earlier, we usually have $k < |Q|$ because we want to choose a small number k of users that have the required skills Q . Note that this is a challenging problem and our main focus in this work; in the case of $k \geq |Q|$, the attribute covering constraint is easily satisfiable.

Observe that according to the problem definition, we look for the seed set with exactly k nodes and an empty set will be returned if such k nodes cannot be found, even if a smaller size seed set can already cover all the attributes in the query.

Theorem 14: The ICS problem is NP-hard. $\square$

Proof: The traditional influence maximization problem is NP-hard [KKT03b]. It can be regarded as a special case of ICS problem that the attribute set of each node can always cover the query Q . Hence the ICS problem is NP-hard.

Remark Our solutions to the ICS problem are orthogonal to the influence probability of each edge in $\mathcal{G}$, whose interpretation is an application-specific issue [KKT03b] and is beyond the scope of this work. We assume that we have influence probability of each edge in $\mathcal{G}$. In practice, there are different ways to set the probability [GBL11, BBM12, KKT03b, JHC12a]. For example, we can adopt the approach [BBM12] to learn a set of topic-aware influence probabilities for each edge, from which we determine the probability of each edge for an attribute set Q based on the relevance between query Q and the topics. Alternatively, we can follow other previous work on influence maximization (*e.g.*, [KKT03b, JHC12a]).

6.2 Greedy solutions

In this section, we present two greedy algorithms for mining influential cover set (ICS).

6.2.1 Score-based Greedy Algorithm

To select seeds in a greedy manner, we need to take both attribute coverage and influence spread into consideration. Specifically, the node that we greedily select in each iteration is supposed to have a high marginal influence increase and cover many uncovered attributes.

To achieve this, we introduce a *ranking function* to measure the score of each node, which will be used to select nodes in the greedy algorithm. Given a set of uncovered attributes Q' and a set of selected seed nodes S' , the final score of node v with respect to S' and Q' is a linear interpolation of two factors, namely the score for attribute coverage and the score for influence spread

$$Score(v, S', Q') = \alpha \frac{|\mathcal{F}(v, Q')|}{\max_{u \in V_Q} |\mathcal{F}(u, Q')|} + (1 - \alpha) \frac{\sigma_{\mathcal{G}}(\{v\}, S')^+}{\max_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\}, S')^+} \quad (6.2)$$

where $\alpha \in [0, 1]$ is a parameter used to balance them, $\mathcal{F}(v, Q') = \mathcal{A}(v) \cap Q'$ is the set of attributes that are newly covered by v , and $\sigma_{\mathcal{G}}(\{v\}, S')^+$ is the marginal influence increase of node v with respect to S' . Note that if $\max_{u \in V_Q} |\mathcal{F}(u, Q')|$ (resp. $\max_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\}, S')^+$) is 0, then $\frac{|\mathcal{F}(v, Q')|}{\max_{u \in V_Q} |\mathcal{F}(u, Q')|}$ (resp. $\frac{\sigma_{\mathcal{G}}(\{v\}, S')^+}{\max_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\}, S')^+}$) is set as 1.

Based on the above score, we develop a greedy algorithm to select seed nodes iteratively, called **ScoreGreedy**. It selects seeds in k iterations. In each iteration, the algorithm selects the node with the maximum score and includes it in the seed set. If all attributes

Algorithm 12: PigeonGreedy

Input: $Q, k, \mathcal{G}$
Output: Seed Set S

```

1  $S \leftarrow \emptyset, Q' \leftarrow Q;$ 
2 while  $|S| < k$  do
3    $L \leftarrow \{u \mid |\mathcal{A}(u) \cap Q'| \geq \lceil \frac{|Q'|}{k-|S|} \rceil, u \in V_Q\} - S;$ 
4   if  $L = \emptyset$  then return  $\emptyset;$ 
5   ;
6    $v \leftarrow \arg \max_{u \in L} \sigma_{\mathcal{G}}(\{u\}, S)^+;$ 
7    $Q' \leftarrow Q' - \mathcal{A}(v); S \leftarrow S \cup \{v\}$ 
8 return  $S;$ 
    
```

are covered by the k selected nodes, **ScoreGreedy** returns seed set S as the result; otherwise it returns $\emptyset$.

Note that **ScoreGreedy** cannot guarantee to find the seed set that can cover all attributes even if such a set exists.

Time complexity. Let $t(n)$ denote the time complexity of evaluating the influence spread of a node set in G , where n is the number of nodes in G . The time complexity of **ScoreGreedy** is $O(k \cdot |V_Q| \cdot t(n))$.

6.2.2 PigeonGreedy Algorithm

We propose another greedy algorithm that leverages the pigeonhole principle to select seed nodes in a greedy manner.

Lemma 23: If a seed set S with k nodes can cover the attribute set Q , then at least one node in S can cover no fewer than $\lceil \frac{|Q|}{k} \rceil$ attributes. $\square$

Based on Lemma 23, we propose the **PigeonGreedy** algorithm outlined in Algorithm 12. Algorithm 12 follows the pigeonhole principle. The main idea is to iteratively apply Lemma 23 in a greedy manner. **PigeonGreedy** computes a set of nodes L which can cover more than $\lceil \frac{|Q'|}{k-|S|} \rceil$ uncovered attributes, where Q' is the set of uncovered attributes. Then it selects the node with maximum marginal influence increase from L . The algorithm invokes this procedure iteratively until k nodes are added to the seed set.

Time complexity. Let n_i be the number of nodes in L when selecting the i -th seed node. Let $t(n)$ denote the complexity of evaluating influence spread for one set in G , where n is the number of nodes in G . Therefore, the time cost of **PigeonGreedy** is $O(t(n)(\sum_{i \in [1, k]} n_i))$.

Theorem 15: **PigeonGreedy** can always return a seed set consisting of k nodes that covers attributes in the query if such a set exists when $k \geq |Q|$. $\square$

Figure 6.1: An example of the ICS problem.

Proof: When $k \geq |Q|$, $\lceil \frac{|Q|}{k-|S|} \rceil = 1$. So we can cover at least one attribute in each iteration. Hence **PigeonGreedy** can always return a seed set satisfying the constraint if such a set exists.

Remark. Though **PigeonGreedy** can guarantee to return a seed set covering all attributes if such a set exists when $k \geq |Q|$, it does not hold when $k < |Q|$. We illustrate this with an example.

Example 27: Consider the graph in Figure 6.1. Let $Q = \{a, b, c, d, e, f\}$. Let $\mathcal{A}(v_1) = \{a, f\}$, $\mathcal{A}(v_2) = \{a, b, c, d\}$, $\mathcal{A}(v_3) = \{a, b, c, d, e\}$, and $\mathcal{A}(v_4) = \{b, c, d\}$. Suppose a user wants to select 2 nodes to cover all the attributes in Q . When **PigeonGreedy** is employed, we first select a node with maximum influence from the nodes that can cover no less than $\frac{6}{2} = 3$ attributes. Assume that v_2 produces larger influence spread than v_3 . So we add v_2 to the seed set. Now $\{e, f\}$ are still uncovered and we can only choose one more node. But there exists no node that can cover both $\{e, f\}$. Hence the algorithm returns an empty set. However, there exists a seed set of 2 nodes, $\{v_1, v_3\}$, which cover all attributes in Q . $\square$

The two algorithms proposed here exploit different greedy strategies. As we shall see later, **ScoreGreedy** usually has a better performance when k is small while **PigeonGreedy** can usually find a seed set with larger influence when k is large. However, neither of them can guarantee to return a seed set if such a set exists.

6.3 Approximation Solutions

In this section, we present the *Partition-based Influential Cover Set* (PICS) algorithm which addresses the limitation of the aforementioned greedy approaches by guaranteeing finding a seed set that covers all attributes in a query if such seed set exists. Furthermore, to reduce unnecessary computation in PICS, we propose an optimized variant of PICS which we refer to as PICS+.

6.3.1 Terminology

We first introduce some terminologies to facilitate exposition.

Algorithm 13: PICS

Input: $Q, k, \mathcal{G}$
Output: Seed Set S

```

1  $S \leftarrow \emptyset$ ;
2 if  $|V_Q| < k$  then return  $\emptyset$  ;
3 for  $(j \leftarrow 0; j < k; j \leftarrow j + 1)$  do
4    $SP[j] \leftarrow \arg \max_{u \in V_Q - S'} \sigma_{\mathcal{G}}(u, S')^+$ ;
5    $S' \leftarrow S' \cup \{SP[j]\}$  ;
6 for each partition  $P = \{A_1, \dots, A_m\}$  of  $Q$  do
7    $S_{\bar{P}} \leftarrow \emptyset; j \leftarrow 0$ ;
8   while  $|S_{\bar{P}}| < k - |P|$  do
9     For each  $A_i \in P$ , remove attributes covered by  $S_{\bar{P}}$ , and remove  $A_i$  from  $P$  if all
       attributes in  $A_i$  are covered;
10     $S_{\bar{P}} \leftarrow S_{\bar{P}} \cup \{SP[j]\}; j \leftarrow j + 1$ ;
11    if  $\sigma_{\mathcal{G}}(S_{\bar{P}}) + \text{upper}(P) > \sigma_{\mathcal{G}}(S)$  then
12       $S_P \leftarrow \text{PartitionSelect}(S_{\bar{P}}, \mathcal{G}, \{V(A_{i_1}), \dots, V(A_{i_h})\})$ ;
13      if  $\sigma_{\mathcal{G}}(S_P \cup S_{\bar{P}}) > \sigma_{\mathcal{G}}(S)$  then
14         $S \leftarrow S_P \cup S_{\bar{P}}$ ;
15 return  $S$ ;

```

Definition 28: Partition. Given a set Q of attributes and a positive integer k , $P = \{A_1, \dots, A_m\}$ ($0 \leq m \leq k$) is a **partition** of Q if and only if

- (1) for each $i \in [1, m]$, attribute set $A_i \subseteq Q$ is not empty;
- (2) for any $i \neq j$ ($i, j \in [1, m]$), $A_i \cap A_j = \emptyset$.
- (3) $\bigcup_{i \in [1, m]} A_i = Q$, i.e., $A_1, \dots, A_m$ together cover Q .

We call $\bigcup_{A_i \in P} A_i$ the **attributes** of P . Given two sets of attributes Q_1 and Q_2 , $Q_1 \subseteq Q_2$, and a **partition** P' of Q_1 , we say that P' is a **partial partition** of Q_2 . $\square$

Example 28: Consider an attribute set $Q = \{a, b, c, d, e\}$ and an integer $k = 3$. Then we have the following: (1) $P_1 = \{\{a, b, c\}, \{d, e\}\}$ is a **partition**; (2) $P_2 = \{\{b, c, d\}, \{a, e\}\}$ is a **partition**; (3) $P_3 = \{\{a, b, c\}, \{c, d\}, \{e\}\}$ is not a **partition** since $\{a, b, c\} \cap \{c, d\} \neq \emptyset$; (4) $P_4 = \{\{a, b\}, \{d, e\}\}$ is not a **partition** since c is not covered by P_4 ; And (5) $P_5 = \{\{a, b\}, \{c\}, \{d\}, \{e\}\}$ is not a **partition** since P_5 contains more than 3 attribute sets. $\square$

Lemma 24: For a set Q of attributes and a positive integer k , the number of all possible **partitions** of Q is bounded by $\sum_{i=1}^{|Q|} \{i^{|Q|}\}$, where $\{i^{|Q|}\}$ is a stirling number of the second type, i.e. $\{i^{|Q|}\} = \frac{1}{i!} \sum_{j=0}^i (-1)^{i-j} \binom{i}{j} j^{|Q|}$. $\square$

Proof: There are two cases to consider: $k \leq |Q|$ or $k > |Q|$.

(1) When $k \leq |Q|$, it is easy to know that there are $\sum_{i=1}^k \binom{|Q|}{i} \leq \sum_{i=1}^{|Q|} \binom{|Q|}{i}$ **partitions**. Recall that the stirling number of the second type $\binom{|Q|}{i}$ is the number of ways to partition a set of $|Q|$ elements into i nonempty subsets [GKP94].

(2) When $k > |Q|$, there are at most $|Q|$ attribute sets in a **partition** by its definition. Then similar to (1), we know that there are $\sum_{i=1}^{|Q|} \binom{|Q|}{i}$ possible **partitions**.

By (1) and (2) above, we have the conclusion.

6.3.2 The PICS Algorithm

We now give the main intuition behind PICS. Recall an attribute set A_i in a **partition** P corresponds to a set of nodes $V(A_i)$ that each contains A_i . Clearly, if for every attribute set A_i of **partition** P , we select a node from $V(A_i)$ to form a seed set, it is guaranteed that such a seed set covers all attributes in Q . Since the number $(|P|)$ of attribute sets of P may be less than k , to get a seed set with k nodes, we may need to further select nodes from V_Q . Hence, in PICS, we first select nodes from V_Q since $V(A_i) \subseteq V_Q$, and thus the nodes selected from V_Q are more likely to provide a better influence spread. Then the attributes covered by the nodes selected from V_Q can be removed from P because they have already been covered; we select nodes from $V(A_i)$ for each remaining A_i in P . Among the seed sets selected from every **partition**, we choose a seed set with the maximum influence spread.

More specifically, for each **partition** P , PICS aims to find a seed set with the maximum influence spread that covers all attributes in Q as follows: (1) We greedily select a set of $k - |P|$ nodes, denoted by $S_{\bar{P}}$, from V_Q (without any constraint of attribute coverage); (2) For each attribute set $A_i \in P$, we remove from it the attributes that have been covered by a node in $S_{\bar{P}}$; If all attributes in A_i have been covered, we remove A_i from P , and greedily choose a node from V_Q and add it to $S_{\bar{P}}$; We repeat this step until $|S_{\bar{P}}| + |P| = k$ and no attribute in P is covered by $S_{\bar{P}}$; and (3) The set of remaining attribute set actually forms a partial **partition** of Q , denoted by P' . For each attribute set $A_i \in P'$, we greedily select a node v_i from $V(A_i)$ to cover A_i ; these selected nodes form a set, denoted by S_P . We call the set $S_{\bar{P}}$ *free set*, and the set S_P *constrained set*. (4) Finally, PICS returns the seed set with the largest influence among the seed sets of all **partitions**.

Algorithm 13 outlines the PICS algorithm. It takes as input a set of attributes Q , an integer k , and a social network $\mathcal{G}$. The result seed set is stored by S , which is initialized by an empty set (line 1). In the rare case that $|V_Q| < k$, it is impossible to find a seed set consisting of k nodes to cover all attributes, and the algorithm returns an empty set as the result (line 2). We greedily select k nodes from V_Q and store them in an array SP (lines 3–5), and then for each **partition** P we use array SP to populate free set $S_{\bar{P}}$, which records the set of nodes selected from V_Q (lines 7–10). For each $A_i \in P$, we remove from it attributes covered by $S_{\bar{P}}$ and remove it from P if all attributes in A_i are covered (line 9),

Function PartitionSelect($S_{\bar{P}}, \mathcal{G}, \{V_1, \dots, V_m\}$)

Output: Seed Set S_P

```

1  $S_P \leftarrow \emptyset$ ;
2 for each  $V_i \in \{V_1, \dots, V_m\}$  do
3   if  $V_i - S_P = \emptyset$  then return  $\emptyset$ ;
4   ;
5    $v \leftarrow \arg \max_{u \in (V_i - S_P)} \sigma_{\mathcal{G}}(\{u\}, S_P \cup S_{\bar{P}})^+$ ;
6    $S_P \leftarrow S_P \cup \{v\}$ 
7 return  $S_P$ ;
```

i.e. we can select a node from V_Q rather than $V(A_i)$. Next, PICS selects a node for each remaining attribute set in **partition** P to ensure that all attributes of Q are covered; all the selected nodes form constrained set S_P . Let $A_{i_1}, \dots, A_{i_h}$ be the remaining attribute sets in P . Before finding S_P , we first apply an optimization (discussed later) to check if **partition** P can be pruned (line 11). If P cannot be pruned, PICS invokes PartitionSelect to find constrained set S_P with the node sets that can cover the remaining attribute sets of **partition** P as the arguments (line 12). Result S is updated if $S_P \cup S_{\bar{P}}$ has a higher influence (lines 13–14). When all **partitions** are considered, S is returned as the final result.

The PartitionSelect procedure takes as input a set of attributes Q , an integer k , a network $\mathcal{G}$, and a group of node sets $\{V_1, \dots, V_m\}$. It first checks whether V_i contains at least one node that has not been added to S_P yet. If not, empty set is returned as the result. Otherwise, it picks the node with maximum marginal influence increase from $V_i - S_P$ (line 4). It repeats this until every node set V_i provides one node to S_P .

Example 29: Reconsider Example 27. Q has many **partitions** and $P_1 = \{\{a, b, c, d, e, f\}\}$ is a **partition**. When processing P_1 in PICS, we add $SP[0]$ to $S_{\bar{P}_1}$ due to $k - |P_1| = 1 > 0$. We assume $SP[0] = v_1$. We then remove attributes of v_1 from each attribute set in P_1 and P_1 becomes $\{\{b, c, d, e\}\}$. Now $k = |P_1| + |S_{\bar{P}_1}|$ and free set $S_{\bar{P}_1} = \{v_1\}$. Assume that $\sigma_{\mathcal{G}}(S_{\bar{P}_1}) + upper(P_1) > \sigma_{\mathcal{G}}(S)$ and we invoke PartitionSelect to compute constrained set S_{P_1} . Now P_1 only contains one attribute set $\{b, c, d, e\}$ and $\{v_3\}$ is the node set that can cover it. PartitionSelect returns $\{v_3\}$ as S_{P_1} . Therefore the seed set for P_1 is $\{v_1, v_3\}$. $\square$

Next, we compute an upper bound of influence spread for any constrained set S_P selected for set P , which can be a **partition** or a partial **partition** (if any A_i is removed from P at line 12).

Definition 29: Upper bound of Influence Spread. The upper bound of P *w.r.t.* $S_{\bar{P}}$ is defined by

$$upper(P, S_{\bar{P}}) = \sum_{A_i \in P} \max_{v \in V(A_i)} \sigma_{\mathcal{G}}(\{v\}, S_{\bar{P}})^+$$

□

Lemma 25: We have $\text{upper}(P, S_{\bar{P}}) \geq \sigma_{\mathcal{G}}(S_P, S_{\bar{P}})^+$ for any S_P returned by function PartitionSelect. □

Proof: Since the function $\sigma_{\mathcal{G}}()$ is submodular, we have

$$\sigma_{\mathcal{G}}(S_P, S_{\bar{P}})^+ \leq \sum_{v \in S_P} \sigma_{\mathcal{G}}(\{v\}, S_{\bar{P}})^+ \leq \sum_{A_i \in P} \max_{v \in V(A_i)} \sigma_{\mathcal{G}}(\{v\}, S_{\bar{P}})^+$$

Based on Lemma 25, if the upper bound together with the influence of $S_{\bar{P}}$ is smaller than the known best result $\sigma_{\mathcal{G}}(S)$, we simply prune the partition P to avoid finding a seed set for it.

Theorem 16: Given a network $\mathcal{G}(V, E, w, \mathcal{A})$, a set of attributes Q , and an integer k , PICS can always find a set of k nodes S that cover all attributes in Q , if such a set exists. □

Proof: Assume that S is a seed set that covers all attributes in Q , which implies $|V_Q| \geq k$. We can find m ($\leq k$) nodes $\{v_1, \dots, v_m\}$ from S such that $\cup_{i \in [1, m]} \mathcal{A}(v_i) = Q$ and for each node v_i there exists an attribute $a_i \in Q$ that is covered by v_i but not other $m - 1$ nodes. The m nodes correspond to a partition $P = \{A_1, \dots, A_m\}$ such that $v_i \in V(A_i)$, i.e. $V(A_i) \neq \emptyset$. Since PICS considers all possible partitions, the partition P is guaranteed to be identified. For the partition P , it is guaranteed that PICS will generate set $S_{\bar{P}}$ due to $|V_Q| \geq k$. Since $V(A_i)$ is not empty for each A_i in P , PICS can select one node from $V(A_i)$. Hence, PICS can find S covering all attributes if such S exists.

Time Complexity. Let $t(n)$ denote the time complexity of evaluating the influence spread of a node set in G , where n is the number of nodes in G . Given a partition $P = \{A_1, \dots, A_m\}$, it takes $O((k)|V_Q| \cdot t(n))$ to select a seed set for a partition. Since there are at most $\sum_{i=1}^{|Q|} \binom{|Q|}{i}$ partitions, the time complexity of PICS is $O((k)|V_Q| \cdot t(n) \cdot \sum_{i=1}^{|Q|} \binom{|Q|}{i})$.

6.3.3 Approximation Ratio Analysis of PICS

When MC simulations are adopted to compute the influence spread, algorithm PICS has the following performance guarantee.

Theorem 17: If MC simulations are adopted to compute the influence spread, algorithm PICS provides an approximation bound of $1/2 - \phi$, where ϕ is an arbitrarily small positive rational number. □

Proof: By Lemma 24, we know that PICS finishes in polynomial time when $|Q|$ is bounded by a constant. We next show PICS has an approximation bound of $1/2 - \phi$.

First let $S^{OPT} = \{s_1, \dots, s_k\}$ be an optimal solution that the set of k seed nodes has the maximum $\sigma_G(S^{OPT})$. Also let **partition** $P_{OPT} = \{A_1, \dots, A_m\}$ ($m \leq k$) such that for each $A_i \in P_{OPT}$, there must exist an $s_i \in S^{OPT}$ such that $s_i \in V(A_i)$, and, moreover, $s_i \neq s_j$ for A_i and A_j ($i \neq j \in [1, m]$). Note that algorithm **PICS** can always find partition P_{OPT} . Let $S = S_{P_{OPT}} \cup S_{\overline{P_{OPT}}}$ be the seed set selected from P_{OPT} by **PICS**, where $S_{P_{OPT}}$ is the constrained set and $S_{\overline{P_{OPT}}}$ is the free set. Without loss of generality, we assume that $S_{P_{OPT}}$ has g nodes, where $g \leq m$. We denote $P' = \{A'_1, \dots, A'_g\}$ as the set of remaining attributes after removing the attributes covered by the free set. Apparently, for each $A'_i \in P'$, we can find a different $A_j \in P_{OPT}$ such that $A'_i \subseteq A_j$ and $V(A_j) \subseteq V(A'_i)$. Thus for each $A'_i \in P'$, we can find a different $s_i \in S^{OPT}$ such that $s_i \in V(A'_i)$. We use $S_{P_{OPT}}^{OPT}$ to denote such nodes and S^{OPT} is divided into $S_{P_{OPT}}^{OPT}$ and $S_{\overline{P_{OPT}}}^{OPT}$ such that $S_{P_{OPT}}^{OPT} = \{s_1, \dots, s_g\}$ and $S_{\overline{P_{OPT}}}^{OPT} = \{u_1, \dots, u_k\}$, where $s_i \in V(A'_i)$ and $u_i \in V(A'_i)$ for any $i \in [1, g]$. We also assume that $S_{\overline{P_{OPT}}}^{OPT} = \{s_{g+1}, \dots, s_k\}$ and $S_{\overline{P_{OPT}}} = \{u_{g+1}, \dots, u_k\}$.

Since the influence function $\sigma_G()$ is submodular and monotone, and $\forall S \subseteq V$, $t \in V$, $\sigma_G(t, S)^+ \geq 0$, we have:

$$\begin{aligned} \sigma_G(S^{OPT}) &\leq \sigma_G(S) + \sum_{s \in S^{OPT} - S} \sigma_G(\{s\}, S)^+ \\ &\leq \sigma_G(S) + \sum_{s_i \in S^{OPT}} \sigma_G(\{s_i\}, S)^+ \end{aligned} \tag{6.3}$$

Let S_i be the set of nodes after the i -th seed node is added to the seed node set. For any node $v \in V$, $\sigma_G(\{v\}, S_i)^+ \geq \sigma_G(\{v\}, S)^+$. It follows that $\sigma_G(S^{OPT}) \leq \sigma_G(S) + \sum_{s_i \in S^{OPT}} \sigma_G(\{s_i\}, S_{i-1})^+$.

When selecting the i -th seed node u_j , there are two cases to consider. (1) When the i -th node is from the free set, **PICS** selects the node with maximum approximate marginal influence increase from V_Q . Then the node that **PICS** selects has a larger influence than any node in $S_{P_{OPT}}^{OPT}$. (2) When the i -th selected node is from the constrained set, it is the node with maximum marginal influence increase in $V(A'_j)$. Since $s_j \in V(A'_j)$, we know that the u_j can produce a higher marginal influence increase than s_j . Thus, for each node u_i in S , there's a different node s_i in S^{OPT} such that u_i has a larger marginal influence increase than s_i . By MC simulations, we can obtain arbitrarily close approximations to the exact influence with a high probability. Hence, for any $\varepsilon > 0$, there is a $\gamma > 0$ such that by using $(1 + \gamma)$ -approximate value of $\sigma_G()$, we have

$$\sigma_G(\{u_j\}, S_{i-1})^+ + \varepsilon \geq \sigma_G(\{s_j\}, S_{i-1})^+ \tag{6.4}$$

By Equation 6.4, we have the following.

$$\begin{aligned}
 \sigma_{\mathcal{G}}(S^{OPT}) &\leq \sigma_{\mathcal{G}}(S) + \sum_{s_i \in S^{OPT}} (\sigma_{\mathcal{G}}(\{s_i\}, S_{i-1})^+ + \varepsilon) \\
 &\leq \sigma_{\mathcal{G}}(S) + \sum_{u_i \in S} \sigma_{\mathcal{G}}(\{u_i\}, S_{i-1})^+ + k\varepsilon \\
 &= \sigma_{\mathcal{G}}(S) + \sigma_{\mathcal{G}}(S) + k\varepsilon = 2 \cdot \sigma_{\mathcal{G}}(S) + k\varepsilon
 \end{aligned} \tag{6.5}$$

That is, $\sigma_{\mathcal{G}}(S) \geq \frac{1}{2} \cdot \sigma_{\mathcal{G}}(S^{OPT}) - \frac{k}{2}\varepsilon$ for both cases.

Let S_{max} be the seed set returned by Algorithm PICS. Since PICS returns the seed set with maximum approximate influence spread, $\sigma_{\mathcal{G}}(S_{max}) \geq \sigma_{\mathcal{G}}(S) - \varepsilon$. Let $\phi = (\frac{k}{2} + 1)\varepsilon$ and ϕ can be an arbitrarily small positive rational value by changing ε . Then we have $\sigma_{\mathcal{G}}(S) \geq \frac{1}{2} \cdot \sigma_{\mathcal{G}}(S^{OPT}) - \phi$.

Put these together, we have the conclusion.

Next we discuss the approximation ratio when using heuristic methods to approximate the influence spread for PICS, such as DegreeDiscount [CWY09], MIP [CWW10b], ISP [LCXZ12], and IRIE [JHC12a]. Though these heuristic methods cannot provide a theoretical bound for influence spread, they perform well in practice in terms of both efficiency and influence spread, and they are also submodular and monotone.

Let $\mathcal{H}_{\mathcal{G}}$ be the class of heuristic algorithms for influence approximation that are submodular and monotone, and let $\hat{\sigma}_{\mathcal{G}} \in \mathcal{H}_{\mathcal{G}}$ be one of the heuristic methods in $\mathcal{H}_{\mathcal{G}}$. Now assume that $\hat{\sigma}_{\mathcal{G}}$ is employed to compute the influence spread. We have the following result.

Theorem 18: $\hat{\sigma}_{\mathcal{G}}(S) \geq 1/2 \hat{\sigma}_{\mathcal{G}}(S_{OPT})$, where S is a seed set returned by PICS based on $\hat{\sigma}_{\mathcal{G}}$, and S_{OPT} is an optimal seed set whose influence $\hat{\sigma}_{\mathcal{G}}(S_{OPT})$ is maximum. $\square$

Proof: Since $\hat{\sigma}_{\mathcal{G}}$ is submodular and monotone, by the proof of Theorem 17 we have

$$\hat{\sigma}_{\mathcal{G}}(S_{OPT}) \leq \hat{\sigma}_{\mathcal{G}}(S) + \sum_{s_i \in S_{OPT}} \hat{\sigma}_{\mathcal{G}}(\{s_i\}, S_{i-1})^+,$$

where S_i is the set of nodes after the i -th seed node is added to the seed set. For $s_i \in S_{OPT} = \{s_1, \dots, s_k\}$ and $u_i \in S = \{u_1, \dots, u_k\}$, we also have $\hat{\sigma}_{\mathcal{G}}(\{s_i\}, S_{i-1})^+ \leq \hat{\sigma}_{\mathcal{G}}(\{u_i\}, S_{i-1})^+$. Therefore, it follows that

$$\begin{aligned}
 \hat{\sigma}_{\mathcal{G}}(S_{OPT}) &\leq \hat{\sigma}_{\mathcal{G}}(S) + \sum_{u_i \in S} \hat{\sigma}_{\mathcal{G}}(\{u_i\}, S_{i-1})^+ \\
 &= \hat{\sigma}_{\mathcal{G}}(S) + \hat{\sigma}_{\mathcal{G}}(S) = 2\hat{\sigma}_{\mathcal{G}}(S)
 \end{aligned} \tag{6.6}$$

6.3.4 Optimized PICS Algorithm

Observe that the PICS algorithm needs to enumerate all **partitions**, which can be inefficient. To alleviate this problem, we leverage the notion of **cover-groups** so that the **partitions** can be reorganized by their **cover-groups**. Consequently, as we shall see later, such organization will enable us to develop a Threshold-Algorithm-flavored [FLN01] approach to avoid unnecessary enumeration of **partitions**. We also show that our optimized algorithm is instance optimal in pruning unnecessary **partitions**.

6.3.4.1 Cover-group

We first introduce the notion of **cover-group** and then elaborate on how we can group **partitions** using **cover-group**.

Definition 30: cover-group. Consider a set of attributes Q and a partial **partition** $P = \{A_1, A_2, \dots, A_m\}$ of Q , and let $r_i = |A_i|$ ($i \in [1, m]$) be the number of attributes of A_i . The multiset $R = \{r_1, r_2, \dots, r_m\}$ is called the **cover-group** of P . $\square$

Example 30: Consider $Q = \{a, b, c, d, e\}$. (1) $P_1 = \{\{c, d\}, \{e\}\}$ is a partial **partition** of Q and its **cover-group** is $\{2, 1\}$. (2) $P_2 = \{\{c, e\}, \{d\}\}$ is also a partial **partition** of Q and its **cover-group** is $\{2, 1\}$. Observe that although P_1 and P_2 are 2 different partial **partitions** of Q , they have the same **cover-group**. $\square$

Recall that for each **partition** P the seed set returned by the PICS algorithm comprises two parts: free set $S_{\bar{P}}$ and constrained set S_P . Nodes in $S_{\bar{P}}$ are selected from V_Q . Nodes in constrained set S_P are selected based on P' , a partial **partition** of Q , which comprises the remaining attribute sets after removing attributes covered by nodes in $S_{\bar{P}}$. Hereafter, when we say partial **partition** of a **partition**, we refer to the partial **partition** like P' .

We group **partitions** in two steps. First, we group **partitions** according to their free sets ($S_{\bar{P}}$), *i.e.* **partitions** with the same free set will be put into the same group. This means that for **partitions** in the same group, their partial **partitions** contain the same number of attribute sets and covers the same set of attributes. We generate at most $\min(k, |Q|)$ groups in this step because there are at most $\min(k, |Q|)$ free set. Second, for **partitions** in each group generated in the first step, we further group them based on their partial **partitions'** **cover-groups**, *i.e.* if their partial **partitions'** **cover-groups** are the same, they form a group.

Observe that each group has a distinct **cover-group** and can be uniquely identified by its **cover-group**, which is shared by all the **partitions** in the group. In addition, for each group R , its **partitions** share the same free set, denoted by $S_{\bar{R}}$. For group R , the set of attributes that are not covered by nodes in $S_{\bar{R}}$ are denoted by Q_R ; they actually form the attribute sets of the partial **partitions** of **partitions** in the group.

Example 31: Consider a set of attributes $Q = \{a, b, c, d, e\}$, $k = 4$ and two partitions of Q , $P_1 = \{\{a, b\}, \{c\}, \{d, e\}\}$ and $P_2 = \{\{a, b, c\}, \{d, e\}\}$. Assume that the first two nodes selected from V_Q can cover $\{c, d\}$. Then for both P_1 and P_2 , their free set is the same, and their partial partition is $\{\{a, b\}, \{e\}\}$. Hence, the two partitions are in the same group. $\square$

Given Q , the number of cover-groups is no more than the number of integer partitions of $|Q|$, which is much smaller than the number of partitions [AS64] (e.g., at $|Q| = 10$, the number of cover-groups is bounded by 42 while the number of partitions can be 115,975). To enumerate all the cover-groups, we enumerate cover-groups for each size. To enumerate cover-groups of size i , we first select $k - i$ nodes from V_Q greedily, which form the free set. Let Q_R be the set of attributes not covered by the $k - i$ nodes in the free set. Attributes in Q_R form partial partitions of size i . Every way of decomposing $|Q_R|$ as the sum of i positive integers corresponds to a cover-group. In case of $Q_R = \emptyset$ and $i = 0$, the only cover-group is $\emptyset$.

6.3.4.2 The Upper bound Algorithm (UA)

Observe that each element $R.r_i$ of cover-group R may correspond to many attribute sets, each of which contains $R.r_i$ number of uncovered attributes and may correspond to many nodes that contain the attribute set. For each element $R.r_i$, we order its corresponding attribute sets in descending order of their influence upper bounds to form a list. This enables us to access partial partitions under the cover-group by scanning these lists, and design an algorithm called *Upper bound Algorithm* (UA) to prune enumerating partial partitions, based on the no random access (NRA) algorithm [FLN01] and compute a seed set for each enumerated partial partition. After we process all cover-groups, the seed set with maximum influence spread is returned as the final result.

We now proceed to detail how to generate lists for cover-group R . Each element $R.r_i$ ($1 \leq i \leq m$) corresponds to a set of attribute sets, each of which contains $R.r_i$ attributes which are not covered by any node in $S_{\bar{R}}$. For each element $R.r_i$, we build a list L_i for its set of attribute sets. Each tuple t in list L_i is associated with three types of information: an attribute set A_t that comprises r_i attributes in Q_R , a set of nodes V_t each of which can cover A_t , and the maximum marginal influence increase of nodes in V_t w.r.t. $S_{\bar{R}}$, i.e. $\max_{v \in V_t} \sigma_{\mathcal{G}}(\{v\}, S_{\bar{R}})^+$. We denote the three types of information of a tuple t by $Attr(t)$, $Node(t)$, and $Inf(t)$, respectively. The tuples in list L_i are ranked in descending order of their maximum marginal influence increase $Inf(t)$. We say $t_1 = t_2$ if and only if (1) $Attr(t_1) = Attr(t_2)$, (2) $Node(t_1) = Node(t_2)$, and (3) $Inf(t_1) = Inf(t_2)$.

Example 32: Consider a network $\mathcal{G}(V, E, w, \mathcal{A})$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, and $\mathcal{A}(v_1) = \{a, b, c\}$, $\mathcal{A}(v_2) = \{a, b, e\}$, $\mathcal{A}(v_3) = \{b, c, f\}$, $\mathcal{A}(v_4) = \{c, d, e\}$, $\mathcal{A}(v_5) = \{d, e\}$, and $\mathcal{A}(v_6) = \{f\}$. Let $Q = \{a, b, c, d, e\}$, $k = 3$, cover-group $R = \{3, 2, 1\}$, and $S_{\bar{R}} = \emptyset$.

Table 6.2: Examples for UA.

$List_1(r_1 = 3)$	$List_2(r_2 = 2)$	$List_3(r_3 = 1)$
$t_1 : [\{abc\}, \{v_1\}, 40]$	$t_2 : [\{de\}, \{v_4, v_5\}, 60]$	$t_3 : [\{f\}, \{v_3, v_6\}, 70]$
$t_4 : [\{abe\}, \{v_2\}, 30]$	$t_5 : [\{ab\}, \{v_1, v_2\}, 40]$	$t_6 : [\{d\}, \{v_4, v_5\}, 60]$
$t_7 : [\{bcf\}, \{v_3\}, 20]$	$t_8 : [\{bc\}, \{v_1, v_3\}, 40]$	$t_9 : [\{e\}, \{v_2, v_4, v_5\}, 60]$
$t_{10} : [\{cde\}, \{v_4\}, 10]$	$t_{11} : [\{ac\}, \{v_1\}, 40]$	$t_{12} : [\{a\}, \{v_1, v_2\}, 40]$
	$t_{13} : [\{ae\}, \{v_2\}, 30]$	$t_{14} : [\{b\}, \{v_1, v_2, v_3\}, 40]$
	$t_{15} : [\{be\}, \{v_2\}, 30]$	$t_{16} : [\{c\}, \{v_1, v_3, v_4\}, 40]$
	$t_{17} : [\{bf\}, \{v_3\}, 20]$	
	$t_{18} : [\{cf\}, \{v_3\}, 20]$	
	$t_{19} : [\{cd\}, \{v_4\}, 10]$	
	$t_{20} : [\{ce\}, \{v_4\}, 10]$	

Let the marginal influence increase of v_1, v_2, v_3, v_4, v_5 w.r.t. $S_{\bar{R}}$ be 40, 30, 20, 10, 60, 70, respectively. We construct $List_1, List_2, List_3$ as shown in Table 6.2. $\square$

We can also divide a tuple t into several smaller tuples $t_1, t_2, \dots, t_m$, where (a) $Attr(t_i) = Attr(t)$ ($1 \leq i \leq m$), (b) $\cup_{i \in [1, m]} Node(t_i) = Node(t)$, and (c) $Inf(t_i)$ is computed within its $Node(t_i)$.

Next, based on the lists of a **cover-group** R , we present our NRA-inspired UA approach to pruning the enumeration of **partitions** in R while finding the **partition** with the maximum influence. Note that NRA, which is proposed to solve the well-known top- k problem, cannot be directly adopted here. Specifically, our goal is to prune the enumeration of **partition** without missing the result. In contrast to the setting for finding top- k objects, a tuple in a list L_i can be combined with multiple tuples in any other list L_j ($j \neq i$) of R to cover Q_R . Hence, the NRA algorithm (and its variants) cannot be used directly. To facilitate the exposition of UA, we first define the notion of **combination**.

Definition 31:Combination. Given a **cover-group** R and a set of tuple lists $\{L_1, \dots, L_{|R|}\}$, we call a group of p ($p \leq |R|$) tuples $t_1, \dots, t_p$, each from a distinct list in $\{L_1, \dots, L_{|R|}\}$, a **combination**, denoted by $\mathcal{T}$, if $Attr(t_i) \cap Attr(t_j) = \emptyset$ for any $i, j \in [1, p], i \neq j$. A combination $\mathcal{T}$ is called a **full combination** if $|\mathcal{T}| = |R|$. $\square$

We say two combinations $\mathcal{T}_1 = \mathcal{T}_2$ if for each tuple t_i in $\mathcal{T}_1$, there exists a tuple t_j in $\mathcal{T}_2$ such that $t_i = t_j$.

Example 33: Consider Table 6.2. Here (1) $\{t_1, t_2\}$ is a **combination**, (2) $\{t_1, t_5\}$ is not a **combination** because $Attr(t_1) \cap Attr(t_5) = \{ab\}$, (3) $\{t_1, t_4\}$ is not a **combination** because the tuples come from a single list $List_1$, and (4) $\{t_1, t_2, t_3\}$ is a **full combination**. $\square$

Our algorithm UA does sorted access in parallel to each of the $|R|$ sorted list L_i . At each depth d (when d tuples have been accessed under sorted access in each list), we

maintain (1) all the generated **combinations**; (2) the influence values associated with the tuples at depth d in the lists; and (3) the upper bound of influence for each **combination** $\mathcal{T}$. Note that if two **combinations** are equal, we only maintain one of them. For a tuple from a list L_i ($1 \leq i \leq m$) at depth d , we check whether it can be combined with any existing **combination** to form a new **combination**.

Once a **combination** becomes a full **combination**, $Attr(t)$ for all its tuple t together forms a partial **partition** of Q_R . We then compute a upper bound influence value using Definition 29. If the upper bound is better than the best seed set so far, we compute the seed set for the combination to check if it has a larger influence, and prune it, otherwise. For each **combination**, we maintain the upper bound of its marginal influence increase using Definition 32.

Definition 32:Upper Bound for combination. Given $R, S_{\bar{R}}$, and depth d , let $x_1^d, \dots, x_m^d$ be the influence values associated with the tuples at depth d in lists $L_1, \dots, L_m$, respectively. Consider a **combination** $\mathcal{T}$, and assume that $L_{i_1}, \dots, L_{i_j}$ ($j = |R| - |\mathcal{T}|$) are the set of lists from which we have not selected any tuple for $\mathcal{T}$. The upper bound marginal influence increase w.r.t. $S_{\bar{R}}$ of $\mathcal{T}$ at depth d is $upper(\mathcal{T})^d = \sum_{t \in \mathcal{T}} Inf(t) + x_{i_1}^d + \dots + x_{i_j}^d$. $\square$

Example 34: Consider the lists in Table 6.2. Let $\mathcal{T} = \{t_1, t_2\}$ and $List_3$ be the list from which no tuples are selected for $\mathcal{T}$. Assume that we access tuples at depth 4. Then we have $x_3^4 = 40$. Hence $upper(\mathcal{T})^4 = inf(t_1) + inf(t_2) + x_3^4 = 40 + 60 + 40 = 140$. $\square$

If the upper bound of marginal influence increase of $\mathcal{T}$ is smaller than the currently best marginal influence increase, we can safely prune the **combination**, *i.e.* any full **combination** extended from it cannot provide a better result. The drop condition is given in Definition 33. When no **combination** is left, our algorithm UA halts.

Definition 33:Drop-condition. Given a set of lists $L_1, \dots, L_m$, a set of nodes $S_{\bar{R}}$, and a **combination** $\mathcal{T}$, let Inf_{max} be the currently best influence spread of any known seed set. We drop $\mathcal{T}$ at depth d if either of the following conditions is satisfied:

1. $upper(\mathcal{T})^d + \sigma_{\mathcal{G}}(S_{\bar{R}}) < Inf_{max}$; or
2. There exists a list L_i from which we have not selected any tuple for $\mathcal{T}$ such that the end of L_i is reached at depth d .

$\square$

Example 35: Consider the lists in Table 6.2. Suppose $S_{\bar{R}} = \emptyset$, $\sigma_{\mathcal{G}}(S_{\bar{R}}) = 0$, and the currently best influence spread of known seed sets is 165. (1) Assume that we access tuples at depth 3, and $\mathcal{T}_1 = \{t_1, t_2\}$ is a **combination**. Its upper bound influence at depth 3 is $upper(\mathcal{T}_1)^3 = Inf(t_1) + Inf(t_2) + x_3^3 = 40 + 60 + 60 = 160$. It is smaller than the

Algorithm 14: UA

Input: $\mathcal{G}, S_{\bar{R}}, L_1, \dots, L_m, S$
Output: Seed set S

```

1   $d \leftarrow 0, C \leftarrow \{\emptyset\};$ 
2  repeat
3      for each  $\mathcal{T} \in C$  do
4          if  $\mathcal{T}$  satisfies the drop-condition then
5              remove  $\mathcal{T}$  from  $C$ ;
6      for tuple  $t_i^d$  in each list  $L_i$  at depth  $d$  do
7          for each  $\mathcal{T} \in C$  do
8               $\mathcal{T}' = \mathcal{T} \cup \{t_i^d\};$ 
9              if  $\mathcal{T}'$  is a full combination then
10                  $S_{\mathcal{T}'} \leftarrow \text{PartitionSelect}(S_{k-m}, \mathcal{G}, \text{Nodes}(\mathcal{T}'));$ 
11                 if  $\sigma_{\mathcal{G}}(S_{\mathcal{T}'} \cup S_{\bar{R}}) > \sigma_{\mathcal{G}}(S)$  then
12                      $S \leftarrow S_{\mathcal{T}'} \cup S_{\bar{R}};$ 
13             else
14                  $C \leftarrow C \cup \{\mathcal{T}'\};$ 
15       $d \leftarrow d + 1;$ 
16 until  $C = \emptyset;$ 
17 return  $S$ 

```

known best influence 165, and thus we drop $\mathcal{T}_1$ at depth 3. (2) Assume that we access tuples at depth 5, and $\mathcal{T}_2 = \{t_2, t_3\}$ is a combination. The end of list $List_1$ is reached at depth 5, and thus $\mathcal{T}_2$ can be dropped. $\square$

Algorithm 14 presents the pseudo code of UA. It takes as input (1) network $\mathcal{G}$, (2) the node set $S_{\bar{R}}$ consisting of $k - |R|$ nodes selected from V_Q , (3) the set of lists $L_1, \dots, L_{|R|}$, and (4) the seed set S with maximum influence spread known so far. In UA, we use the following variables: d indicates the depth of lists that UA is accessing; and C keeps track of all active combinations. At depth d , we first check each combination in C and remove it from C if it satisfies the drop-condition (line 3–5). Then for tuple t_i^d in each list L_i , UA checks each combination in C (lines 6–14). Specifically, it adds t_i^d to each combination to get a new combination (line 8). If a new combination is a full combination, then UA invokes **PartitionSelect** to compute $S_{\mathcal{T}'}$, where argument $\text{Nodes}(\mathcal{T})$ represents the set of node sets for combination $\mathcal{T}$, each node set corresponding to $\text{Nodes}(t)$, $t \in \mathcal{T}$; Otherwise, we add $\mathcal{T}'$ to C (line 14). We update the best known seed set S if the influence of $S_{\mathcal{T}'} \cup S_{\bar{R}}$ is larger than $\sigma_{\mathcal{G}}(S)$ (lines 11–12). We increase d to access tuples at next level (line 15) until C is empty. Finally, we return S .

Example 36: Consider UA runs on the three lists in Table 6.2: Initially, C comprises an $\emptyset$. At depth 1, it computes upper bound influence for $\emptyset$, representing a combination, which is $40 + 60 + 70 = 170$. (1) Assume that the currently known best influence $\sigma_{\mathcal{G}}(S)$ is 180.

UA first remove $\emptyset$ from C because of the drop-condition and then halts. (2) Assume that $\sigma_G(S)$ is 150. UA will find 7 combinations: $\mathcal{T}_1 = \{t_1\}$, $\mathcal{T}_2 = \{t_2\}$, $\mathcal{T}_3 = \{t_1, t_2\}$, $\mathcal{T}_4 = \{t_3\}$, $\mathcal{T}_5 = \{t_1, t_3\}$, $\mathcal{T}_6 = \{t_2, t_3\}$ and $\mathcal{T}_7 = \{t_1, t_2, t_3\}$. Since $\mathcal{T}_7$ is a full combination, UA invokes **PartitionSelect** to compute a seed set for $Nodes(\mathcal{T}_7)$. Assume the seed set is $\{v_1, v_5, v_6\}$ and its influence is 165. We update S and now $\sigma_G(S) = 165$. Then at depth 2, UA next checks the combinations in C . At depth 2, $upper(\emptyset)^2 = 130$, $upper(\mathcal{T}_1)^2 = 140$, $upper(\mathcal{T})^2 = 150$, $upper(\mathcal{T}_3)^2 = 160$, $upper(\mathcal{T}_4)^2 = 140$, $upper(\mathcal{T}_5) = 150$, $upper(\mathcal{T}_6) = 160$. The upper bound of every combination in C is smaller than the best known influence 165, and thus all of them are dropped at depth 2 and C is empty. Therefore the algorithm terminates. $\square$

We next consider the optimality of Algorithm UA.

Theorem 19: Given a cover-group R and a set of nodes $S_{\bar{R}}$, let $\mathbb{D}$ be the class of all generated $|R|$ lists $L_1, \dots, L_{|R|}$ for R . Let $\mathbb{A}$ be the class of correct algorithms that run on every $\{L_1, \dots, L_m\}$ using sorted access to deterministically find the full combination with maximum marginal influence increase with respect to $S_{\bar{P}}$, which is computed by **PartitionSelect**. Apparently, algorithm UA is in $\mathbb{A}$. We say UA is instance optimal over $\mathbb{A}$ and $\mathbb{D}$, i.e. for any algorithm $\mathcal{A} \in \mathbb{A}$ and any $\{L_1, \dots, L_m\} \in \mathbb{D}$, $cost(\text{UA}, \{L_1, \dots, L_m\}) \leq c \times cost(\mathcal{A}, \{L_1, \dots, L_m\})$, where $cost(\text{UA}, \{L_1, \dots, L_m\})$ (resp. $cost(\mathcal{A}, \{L_1, \dots, L_m\})$) is the number of tuples seen by UA (resp. $\mathcal{A}$) and c is a constant. $\square$

Proof: Assume that UA halts at depth d on $D_1 = \{L_1, \dots, L_m\}$. We will show that $\mathcal{A}$ must get to depth d in at least one list in D_1 ; otherwise $\mathcal{A}$ cannot always return the correct answer. For contradiction, we assume that $\mathcal{A}$ can always return the correct answer and it does not get to depth d in any list in D_1 . We will show the contradiction by constructing a $D_2 \in \mathbb{D}$ on which $\mathcal{A}$ cannot find the correct answer.

Since UA halts at depth d on D_1 , there exists a combination $\mathcal{T}$ such that $upper(\mathcal{T})^{d-1} > \sigma_G(S)$. We construct $D_2 = \{L'_1, \dots, L'_m\}$ as follows: L'_i is identical to L_i up to depth $d-1$ (that is, for each $i \in [1, m]$, the top $d-1$ tuples in L'_i are the same as in L_i). Run UA on D_2 . There also exists a combination $\mathcal{T}'$ such that $\mathcal{T}'$ and $\mathcal{T}$ contain the same attribute sets. Let $L'_{j_1}, \dots, L'_{j_x}$ ($x = |R| - |\mathcal{T}'|$) be the set of lists from which we have not selected any tuple for $\mathcal{T}'$. We let the attribute sets in the tuples at depth d in $L'_{j_1}, \dots, L'_{j_x}$ be the particular attribute sets such that $\mathcal{T}'$ can become a full combination $\mathcal{T}_{full}$ if we expand it with these tuples. Let $Inf(t_{j_i})$ for tuple t_{j_i} at depth d in L'_{j_i} be $x_{j_i}^{d-1}$, where $x_{j_i}^{d-1}$ is the influence part of the tuple at depth $d-1$ in list L_{j_i} . So $upper(\mathcal{T}_{full}) > Inf_{max}$. Assume the seed set of $\mathcal{T}$ has a larger marginal influence increase than Inf_{max} . Since the top $d-1$ tuples in each list in D_2 are the same as those in D_1 , and $\mathcal{A}$ does not get to depth d in any lists, the seed sets returned by $\mathcal{A}$ on D_1 and D_2 are the same. So $\mathcal{A}$ cannot find the

Algorithm 15: PICS+

Input: $\mathcal{G}, Q, k$
Output: Seed set S

```

1  $S \leftarrow \emptyset$ ;
2 if  $|V_Q| < k$  then return  $\emptyset$ ;
3 ;
4 foreach  $j$  from 1 to k do
5    $SP[j] \leftarrow \arg \max_{u \in V_Q - S'} \sigma_{\mathcal{G}}(u, S')^+; S' = S' \cup \{SP[j]\}$ ;
6 foreach cover-group  $R$  do
7    $S_{\bar{R}} = \cup_{i \in [1, k - |R|]} \{SP[i]\}$ ;
8   Build lists  $L_1, \dots, L_{|R|}$ ;
9    $S \leftarrow \text{UA}(\mathcal{G}, S_{\bar{R}}, L_1, \dots, L_{|R|}, S)$ ;
10 return  $S$ ;

```

correct answer on D_2 , contradicting that $\mathcal{A}$ is a correct algorithm. Therefore $\mathcal{A}$ must get to depth d in at least one list in D_1 . $\text{cost}(\text{UA}, L_1, \dots, L_m) \leq m \cdot \text{cost}(\mathcal{A}, L_1, \dots, L_m)$, where m is the size of R and is smaller than constant $|Q|$.

6.3.4.3 The PICS+ Algorithm

Finally, we present the PICS+ to process all **cover-groups**, for each of which we use UA to find a seed set for partial **partitions** under the **cover-group** and to update the known best seed set. Note that some **cover-group** may not contribute to the final seed set (*i.e.* it is pruned) as illustrated in Example 36. The pseudo code of PICS+ is shown in Algorithm 15. In PICS+, we use S to record the best known seed set. It first selects k nodes greedily from V_Q and store them in an array SP (lines 3–4). Then for each **cover-group**, it uses array SP to populate $S_{\bar{R}}$, which stores the nodes selected from V_Q (line 6). The algorithm next constructs lists $L_1, \dots, L_{|R|}$ for each element in the **cover-group** R (line 7). It invokes UA to update the best seed set S (line 8). After all **cover-groups** are considered, the algorithm returns S as the final result.

Correctness. This can be verified by showing that the seed set returned by PICS+ is the same as the one returned by PICS.

6.4 Performance Study

6.4.1 Experimental Setup

Datasets. We use 4 datasets in our experiments and their properties are given in Table 6.3. MeetUp is crawled from an event-based social network *meetup.com* from July 2013 to Oct 2013. Each node represents a user and the interests specified by the user

Table 6.3: Dataset properties.

Property	FX	PC	DBLP	MeetUp
# of nodes	38,834	76,665	874,305	1,013,453
# of edges	164,093	1,702,058	9,415,206	34,410,754
# of distinct attr.	37,036	103,289	89,975	64,721
avg. # of attr. per node	47	9	27	11

form the attributes of each node. Two users are connected by an edge if they appear in the same group at least three times. **PC**¹ is a real-life event-based social network from *planecast.com*. Each node represents a user and each edge represents a subscription between two users. We group the events into clusters and the clusters containing the events that a user attended form the attributes of the node. **DBLP** is a real-life network from DBLP. Each node represents an author, and the attributes of a node comprise the words in the author’s publication titles after removing stopping words and stemming. An edge represents the co-author relationship between two authors. Lastly, **FX**² is a dataset of movie ratings from *flixster.com* in 2008. Each node represents a user and the movies that have been rated by the user form the attributes of each user.

Query. A query contains an integer k and a set of attributes Q . We use a pair of integers (x, y) to denote a group of queries where $x = k$ and $y = |Q|$. Attributes in Q are randomly generated such that there exists a set of k seed nodes to cover all attributes in Q . Each group comprises 30 queries.

Evaluated algorithms. The following algorithms are evaluated. (a) **ScoreGreedy** algorithm, denoted by **SG**; (b) **PigeonGreedy** algorithm, denoted by **PG**; and (c) **PICS+** algorithm. Recall that **SG** and **PG** may not always return a seed set that can cover all attributes in the query even if such a set exists. In contrast, **PICS+** guarantees to return a seed set that can cover all attributes in the query if such a set exists. The parameter α in **SG** is set at 0.6 since it usually achieves the best *success rate* (defined below) according to our experiments. We use a state-of-the-art method, **IRIE** [JHC12a], to approximate the influence spread. We also evaluate the performance when MC simulation is used to compute the influence.

Note that there is no existing algorithm that addresses the proposed ICS problem. Nevertheless, since one of our goals is to demonstrate that the seed set generated by our proposed techniques cannot be found by existing influence maximization (IM) approaches, we confine ourselves to compare the **PICS+** against a state-of-the-art IM technique, **IRIE**. All algorithms are implemented in C++. All experiments are run on a Windows PC with Intel Xeon 2.66 GHZ 6-core CPU and 24 GB memory.

¹<http://lsna2012.net76.net/ebsn/>

²<http://www.cs.sfu.ca/~sja25/personal/datasets/>

Figure 6.2: Influence spread under influence probabilities generated by four methods.

Performance measures. We consider three measures. (a) The *success rate* $= a/b$, where b is the number of queries in a group, and a is the number of queries in the group for which an algorithm is able to find a seed set (satisfying attribute constraints). (b) The influence spread of the seed set. We apply 20,000 Monte Carlo simulations and the average number of influenced nodes is used as the influence spread of the seed set. (c) The runtime of algorithms.

Propagation probability. As remarked earlier, our solutions to the ICS problem are orthogonal to the techniques for generating influence probability. Hence, we consider four methods of generating propagation probability. (a) **Degree.** The probability on edge (u, v) is set to be $\frac{1}{N_{in}(v)}$, where $N_{in}(v)$ is the in-degree of v . (b) **Random.** The probability on edge (u, v) is randomly selected from $\{0.1, 0.01, 0.001\}$. (c) **TopicPP.** We first compute a $p_{u,v}$ for each edge (u, v) as $p_{u,v} = \max(\frac{|\mathcal{A}(u)| \cdot |\mathcal{A}(v)| \cdot |\mathcal{A}(u) \cap \mathcal{A}(v)|}{|Q|^3}, 0.001)$. Then the probability on edge (u, v) is set to be $\frac{p_{u,v}}{\sum_{(s,v) \in E} p_{s,v}}$. (d) **TIC.** We adopt the TIC model proposed in [BBM12] and learn the propagation probability from the action log of FX. For each movie a , we can compute the probability $p_{u,v}^a$ that u activates v according to the model. Given a query $Q = \{a_1, \dots, a_q\}$, the weight of edge (u, v) is defined as

Table 6.4: Comparison with an IM technique: Jaccard similarity (JC) and attribute coverage rate (ACR).

Dataset		(2,4)	(3,6)	(4,8)	(5,10)	(6,12)
FX	JC	0.033	0.02	0.01	0.011	0.021
	ACR	3.3%	1.1%	0.8%	2.3%	1.7%
PC	JC	0.033	0.033	0.040	0.054	0.031
	ACR	2.5%	1%	12.5%	10%	0.6%
DBLP	JC	0.044	0.033	0.056	0.079	0.096
	ACR	25.8%	33.9%	31.3%	34%	37.2%
MeetUp	JC	0.0	0.0	0.005	0.0	0.0
	ACR	0%	0.5%	3%	1%	0.8%

$1 - \prod_{a_i \in Q} (1 - p_{u,v}^{a_i})$. Note that the remaining three datasets do not provide historical action logs information.

We study the effects of the propagation probability generated by four methods on influence spread. We run 5 groups of queries, (2, 4), (3, 6), (4, 8), (5, 10), and (6, 12). Figure 6.2 shows the average influence spreads of the three evaluated algorithms on FX. We can see that the relative performance of the four algorithms in terms of influence spread is consistent over the different methods of generating propagation probabilities. The relative performance of the four algorithms in terms of success rate and efficiency is also consistent, irrespective of methods of generating the propagation probability.

We also investigated the impact of adjusting the propagation probability between two nodes based on their degrees. Our results show that it does not significantly affect the relative performances of the algorithms. We omit the detailed results due to space constraint. *In the sequel, the propagation probability is generated by TopicPP.*

6.4.2 Experimental Results

Comparison with im techniques. First, in this set of experiments we compare the seed sets generated by our algorithm (PICS+) and a traditional IM technique (we use the IRIE method) for each query. Specifically, we measure the (1) Jaccard similarity between the seed sets returned by these two techniques, (2) the *attribute coverage rate* of the seed sets returned by traditional IM technique, which is defined as $\frac{|\bigcup_{s \in S} \mathcal{A}(s) \cap Q|}{|Q|}$, and (3) the runtime of these two techniques. Table 6.4 reports the results. Observe that there is very low similarity between the seed sets returned by these two techniques. We can also observe that only a small portion of attributes in the query can be covered by the seed sets returned by the traditional IM technique. Hence we conclude that traditional IM techniques are not suitable for addressing the ICS problem. The runtime of IRIE and PICS+ is reported in Figure 6.3. As expected, IRIE is faster because it does not take into account the attribute covering constraint.

Figure 6.3: Runtime vs. query groups.

Overall Performance. Next, we evaluate the performance of the three proposed methods. We run 5 groups of queries, (2, 4), (3, 6), (4, 8), (5, 10), and (6, 12) for each dataset. Table 6.5 shows the success rates of SG and PG in finding a seed set covering all the attributes in a query for each query group containing 30 queries. Note that PICS+ guarantees to find such a seed set, and thus its success rate is 1. The success rates of SG and PG often are around 0.5, which means they fail on half of 30 queries in a group. On some query groups, the success rate of PG can be as low as 0.17. This indicates that the greedy algorithms may not be a good choice for the ICS problem, especially when k is smaller than $|Q|$.

Figure 6.4 shows the influence spread of the three algorithms on PC, DBLP and MeetUp (Figure 6.2 gives results on FX). If SG or PG cannot find a seed set to cover attributes in some query, it returns an empty set whose influence spread is 0. We can see that the influence spread of PICS+ is consistently better than the other two methods.

Figure 6.3 shows the runtime of different algorithms. Although PICS+ can finish in reasonable time for the largest dataset, it is less efficient than SG or PG as expected.

Table 6.5: Success rate of SG and PG.

Dataset	Method	(2,4)	(3,6)	(4,8)	(5,10)	(6,12)
FX	SG	0.87	0.77	0.87	0.83	0.87
	PG	0.77	0.67	0.7	0.57	0.6
PC	SG	0.47	0.53	0.53	0.5	0.43
	PG	0.63	0.36	0.33	0.23	0.1
DBLP	SG	0.47	0.37	0.53	0.47	0.5
	PG	0.5	0.17	0.23	0.27	0.27
MeetUp	SG	0.33	0.2	0.13	0.2	0.03
	PG	0.47	0.0	0.03	0.03	0.03

Figure 6.4: The influence spread vs. query groups.

Effect of $|Q|$. This set of experiments is conducted to study the effect of $|Q|$. We set $k = 5$ and vary $|Q|$ from 8 to 15. Table 6.6 shows the success rates of SG and PG. We observe that SG and PG may fail to return a seed set for a significant portion of queries. We also observe that the success rate usually drops as $|Q|$ increases, as expected. This is because it is more difficult to cover all the attributes as $|Q|$ increases.

Figure 6.5 shows the influence spread with different $|Q|$. PICS+ outperforms the other two methods consistently; When $|Q|$ gets larger, the disparity between PICS+ and the other two methods become larger. In addition, the influence spread usually drops as $|Q|$ increases on PC, DBLP, and Meetup. This is because when $|Q|$ is small, it is relatively easier to cover the attributes in Q , and thus we have better chances to choose nodes with high influence while satisfying the attribute constraint. On FX, the influence spread is less affected by $|Q|$ since nodes in FX can cover many attributes and it is easier for SG and PG to find a good seed set. Although PICS+ is usually 2-3 times slower than SG and PG, PICS+ still can finish in reasonable time even for the largest network. The runtime of all three methods increases slightly with the increase of $|Q|$. Due to the space limitation, we do not show the runtime.

Effect of k . We evaluate the effect of k by varying it from 5 to 13 while fixing the query attributes in Q and $|Q|$ at 10. As shown in Figure 6.6, PICS+ consistently achieves better influence than the other two methods. However, as k increases, the gap becomes smaller. This is because when k gets larger, it becomes easier to cover all attributes and

Table 6.6: Success rate of SG and PG (Varying $|Q|$).

Dataset	Method	$ Q $							
		8	9	10	11	12	13	14	15
FX	SG	0.83	0.9	0.83	0.93	0.9	0.8	0.83	0.63
	PG	0.63	0.83	0.57	0.8	0.73	0.63	0.57	0.5
PC	SG	0.7	0.6	0.5	0.27	0.33	0.57	0.2	0.37
	PG	0.63	0.7	0.23	0.13	0.2	0.27	0.03	0.13
DBLP	SG	0.77	0.53	0.47	0.23	0.3	0.17	0.23	0.13
	PG	0.67	0.3	0.27	0.2	0.1	0.03	0.1	0.0
MeetUp	SG	0.43	0.26	0.2	0.23	0.33	0.03	0.2	0.1
	PG	0.53	0.26	0.03	0.2	0.2	0.13	0.13	0.0

Table 6.7: Success rate of SG and PG (Varying k).

Dataset	Method	k		
		5	6	7 – 13
FX	SG	0.83	1.0	1.0
	PG	0.56	0.96	1.0
PC	SG	0.5	0.93	1.0
	PG	0.23	0.83	1.0
DBLP	SG	0.47	0.93	1.0
	PG	0.27	0.73	1.0
MeetUp	SG	0.2	0.47	1.0
	PG	0.03	0.6	1.0

the constraints of attributes are weakened. Note that in our problem typically $k < |Q|$ as the number of desired event organizers is small. We also observe that SG outperforms PG when $k < |Q|$, and they perform similarly when $k \geq |Q|$.

PICS+ is usually 2-3 times slower than SG and PG that take similar time. The runtime of PICS+ is almost not affected by k , but SG and PG become slower as k increases. Due to the space limitation, we do not give the runtime results.

The success rate of SG and PG is given in Table 6.7. As expected, the success rate approaches to 1 when the value of k approaches $|Q|$ since it becomes relatively easier to cover attributes in Q . However, we need to temper the observation. Here we use the same set of query attributes for different k values, and each query Q is generated such that Q can be covered in the case of $k = 5$, which means that those queries are easy queries for $k > 5$ (their attributes are easier to be covered for $k > 5$). This also explains why success rate of SG and PG can be 1 at $k = 7$. Nevertheless, the results indicate that when $k \geq |Q|$, SG and PG are a better choice. However, as remarked earlier, k is usually small in our target applications.

Comparison between PICS and PICS+. We compare the efficiency of PICS+ and PICS. Note that they return the same results and thus have the same influence spread (their success rate is 1). Figure 6.9 reports the runtime of the two methods for processing 4 groups of queries, (2, 4), (3, 6), (4, 8), (5, 10) on FX. Note the y-axis is in logarithmic scale.

Figure 6.5: Influence spread vs. $|Q|$ ($k = 5$).

We observe that PICS+ is orders of magnitudes faster than PICS, especially when $|Q|$ becomes larger. The results on the other datasets are qualitatively similar.

Computing Influence by MC Simulations. This experiment is to study whether the relative performance of the three methods is similar when MC Simulations are used to estimate influence. In terms of the success rate, the results using MC simulations are almost the same as those using IRIE reported in Table 6.5. Figure 6.7 shows the runtime and influence spread of the three methods on dataset FX. Experiment for group (6, 12) cannot finish in 5 hours and we terminate it and do not get its runtime. We observe that the relative influence spread of the three methods using MC simulations is similar as that using IRIE. Actually, MC simulations achieve similar influence spread as does IRIE. In terms of runtime, the relative performance of the three methods is also similar; however, by comparing Figure 6.7(a) and Figure 6.3(a), we observe that the runtime using MC simulations is much longer than that using IRIE. The results on the other datasets are qualitatively similar.

(a) FX

(b) PC

(c) DBLP

(d) MeetUp

Figure 6.6: Influence spread vs. k ($|Q| = 10$).

Figure 6.7: Computing influence by MC simulations.

Figure 6.8: Runtime vs. graph size.

Figure 6.9: Runtime of PICS and PICS+.

Scalability with graph size. We extract subgraphs from *MeetUp*. On each graph, we run 5 groups of queries, (2, 4), (3, 6), (4, 8), (5, 10), (6, 10). We only report the runtime for (4, 8) in Figure 6.8. The trend on other groups are similar. We find that all the three algorithms scale well with the graph size.

6.5 Summary

The quest for organizing successful social events has become more pressing due to the rapid growth of event-based online social services. In this chapter, we introduce the problem of discovering influential event organizers from an online social network. We formulate it as the problem of mining influential cover set (ICS), which is intractable. Hence, we propose two greedy algorithms, *ScoreGreedy* and *PigeonGreedy*, to find approximate solutions to the problem. Although they are efficient, they have no guarantee to find a feasible solution. This led us to propose an approximation algorithm called *PICS+*

that guarantees to find a feasible solution if any. The experimental results on real-world datasets demonstrate that **PICS+** guarantees to find a feasible solution if any within reasonable time. Although the two greedy algorithms run faster, they fail to find feasible solution for a significant portion of queries.

Chapter 7

Conclusions and Future Work

In this dissertation, we propose and investigate four group optimization queries on spatial data and social networks.

7.1 Conclusions

In Chapter 3, we tackle the task of best aggregation region detection. Specifically, we formulate a general problem called the *best region search* (*BRS*) problem, which aims at finding a region of a given size such that the submodular aggregation score of the objects inside the region is maximized. In order to address the *BRS* problem, we propose two novel algorithms. The first algorithm, *SliceBRS*, finds an exact answer to the *BRS* problem. The second algorithm, *CoverBRS*, assumes that slight imprecision to the solution is acceptable and finds an approximate answer to the *BRS* problem bounded by a constant. An experimental study with real-world datasets and applications demonstrates the effectiveness and superiority of the proposed algorithms.

In Chapter 4, we tackle the task of bursty region detection. Specifically, we propose a *continuous bursty region detection* (*SURGE*) problem which utilizes such spatial object stream to detect and maintain *bursty region* of a given size in a specified geographic area in real time. We adopt two sliding windows to model the burstiness of a region. To address the *SURGE* problem efficiently, we propose an exact solution and two approximate solutions. In our exact solution, we design various pruning techniques to reduce the search space. We also propose a lazy search strategy to avoid frequent recomputations. Our approximate solutions can monitor a region whose burst score is bounded by $\frac{1-\alpha}{4}$, where α is a parameter to control the approximate ratio. We further extend these solutions to support detection of top- k bursty regions. Extensive experiments with real-world data are conducted to demonstrate the efficiency and effectiveness of our solutions. Our proposed algorithms outperform the baseline algorithm.

In Chapter 5, we tackle the task of attribute-based similar region detection. In particular, we first define the attribute distribution and attribute distance to evaluate the similarity between two regions *w.r.t.* a specific attribute. Then we formulate the problem as the *attribute-based similar region search* (ASRS) problem, which aims at finding a region of the same size as the query rectangle such that the attribute distance between the two regions is minimized. To solve the ASRS problem efficiently, we propose a grid-based algorithm. Our experiments show that the designed pruning techniques are effective and the proposed algorithm can solve the ASRS problem efficiently.

In Chapter 6, we tackle the task of influential organizers detection. We formulate the *influential cover set* problem, which aims to select k users who can together cover a set of required skills and their influence is maximized. We adopt the independent cascade model to evaluate the influence of users. We prove that the problem is NP-hard. Hence, to address the ICS problem, we propose three algorithms to find approximation solutions to the problem. The first two algorithms are greedy; they run faster, but have no guarantees. The third algorithm is 2-approximate and guarantees to find a feasible solution if any. Our empirical study over several real-world networks demonstrates the superiority of our proposed solutions.

7.2 Future Directions

My research opens several potential directions for my future work.

- For the task of best aggregation region detection, when a user is exploring the data, he may issue a query first. Then based on the returned result, he may revise his query and repeat this procedure until he is satisfied with the final result. Thus, it would be interesting if we can reuse the results from a previous query to speed up the computation for the current query.
- For the task of bursty region detection, it would be interesting if we can relax the spatial constraint. Specifically, the size of the region is specified as part of the input. In some cases, it is hard for the user to specify the most appropriate size. Hence, it would be useful if we can monitor the bursty region with an approximate size.
- In our work on attribute-based similar region detection, the result is returned when the computation is finished. It may take some time before the user can see any result. It would be interesting if we can return an approximate result first, and improve the quality of the result gradually.

- For the task of influential organizers detection, we only consider the attribute constraint currently in the ICS problem. However, in real life, the relationships among the organizers also have a significant impact on the event which is to be organized. Thus, it would be useful if we could consider the relations among the selected users. For example, we can consider the distance between any two selected users. As another example, we can consider the topological structure among the selected users.

- **Kaiyu Feng**, Tao Guo, Gao Cong, Sourav S Bhowmick, Shuai Ma. SURGE: Continuous Detection of Bursty Regions Over a Stream of Spatial Objects. (Under Submission)
- Tao Guo, **Kaiyu Feng**, Gao Cong, Zhifeng Bao, Efficient Selection of Geospatial Data on Maps for Interactive and Visualized Exploration. (Under Submission)
- **Kaiyu Feng**, Kaiqi Zhao, Yiding Liu, Gao Cong. A System for Region Search and Exploration (Demo). Proceedings of the VLDB Endowment (**PVLDB 2016**). Volume 9 Issue 13, September 2016, Pages 1549-1552
- **Kaiyu Feng**, Gao Cong, Sourav S Bhowmick, Wen-Chih Peng, Chunyan Miao. Towards Best Region Search for Data Exploration. Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (**SIGMOD 2016**), pages 1055-1070, San Francisco, California, USA, June 26 - July 01, 2016.
- Gao Cong, **Kaiyu Feng**, Kaiqi Zhao. Querying and Mining Geo-textual Data for Exploration: Challenges and Opportunities. 2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW), pages 165-168, Helsinki, Finland, May 16–20, 2016.
- **Kaiyu Feng**, Gao Cong, Sourav S Bhowmick, Shuai Ma. In Search of Influential Event Organizers in Online Social Networks. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (**SIGMOD 2014**), pages 63-74, Snowbird, Utah, USA, June 22–27, 2014.

References

- [ABC⁺12] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. Online team formation in social networks. In *WWW*, 2012.
- [AH16] Daichi Amagata and Takahiro Hara. Monitoring maxrs in spatial data streams. In *EDBT*, pages 317–328, 2016.
- [AHWY03] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the VLDB Endowment*, pages 81–92. VLDB Endowment, 2003.
- [AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 1964.
- [BBCL14] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 946–957, 2014.
- [BBM12] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. In *ICDM*, 2012.
- [BS05] Ahmet Bulut and Ambuj K Singh. A unified framework for monitoring data streams in real time. In *International Conference on Data Engineering*, pages 44–55. IEEE, 2005.
- [CC07] Hyung-Ju Cho and Chin-Wan Chung. Indexing range sum queries in spatio-temporal databases. *Information and Software Technology*, 49(4):324–331, 2007.

REFERENCES

- [CCC13] Lisi Chen, Gao Cong, and Xin Cao. An efficient query indexing mechanism for filtering geo-textual data. In *Proceedings of the SIGMOD*, pages 749–760. ACM, 2013.
- [CCCT15] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. Temporal spatial-keyword top-k publish/subscribe. In *International Conference on Data Engineering*, pages 255–266. IEEE, 2015.
- [CCT12] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. A scalable algorithm for maximizing range sum in spatial databases. *Proceedings of the VLDB Endowment*, 5(11):1088–1099, 2012.
- [Cha86] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.
- [CLW⁺14] Zitong Chen, Yubao Liu, Raymond Chi-Wing Wong, Jiamin Xiong, Ganglin Mai, and Cheng Long. Efficient algorithms for optimal location queries in road networks. In *International Conference on Management of Data, SIGMOD*, pages 123–134, 2014.
- [CM05] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [CR04] Michelangelo Conforti and Romeo Rizzi. Combinatorial optimization - polyhedra and efficiency: A book review. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):153–159, 2004.
- [CWW10a] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 1029–1038, 2010.
- [CWW10b] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. *KDD*, pages 1029–1038, 2010.
- [CWY09] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *SIGKDD*, pages 199–208, 2009.

REFERENCES

- [DGR03] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *Proceedings of the SIGMOD*. ACM, 2003.
- [DZX05] Yang Du, Donghui Zhang, and Tian Xia. The optimal-location query. In *Advances in Spatial and Temporal Databases*, pages 163–180. Springer, 2005.
- [FCLH16] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment*, 9(12):1233–1244, 2016.
- [FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
- [For10] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.
- [FPT81] Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.
- [FYYL05] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. Parameter free bursty events detection in text streams. In *Proceedings of the VLDB Endowment*, pages 181–192. VLDB Endowment, 2005.
- [GBL11] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. A data-based approach to social influence maximization. *VLDB*, 5(1):73–84, 2011.
- [GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 1994.
- [HL17] Xin Huang and Laks VS Lakshmanan. Attribute-driven community search. *Proceedings of the VLDB Endowment*, 10(9):949–960, 2017.
- [HLL⁺15] Huiqi Hu, Yiqun Liu, Guoliang Li, Jianhua Feng, and Kian-Lee Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *International Conference on Data Engineering*, pages 711–722. IEEE, 2015.

REFERENCES

- [HLX17] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. Community search over big graphs: Models, algorithms, and opportunities. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 1451–1454. IEEE, 2017.
- [HTIA17] Muhammed Mas-ud Hussain, Goce Trajcevski, Kazi Ashik Islam, and Mohammed Eunus Ali. Towards efficient maintenance of continuous maxrs query for trajectories. In *EDBT*, 2017.
- [HWQ⁺11] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, and Zhen He. Top-k most influential locations selection. In *Proceedings of the 20th Conference on Information and Knowledge Management*, pages 2377–2380, 2011.
- [IA83] Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of algorithms*, 4(4):310–323, 1983.
- [JHC12a] Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *ICDM*, 2012.
- [JHC12b] Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *International Conference on Data Mining, ICDM*, pages 918–923, 2012.
- [JL98] Marcus Jurgens and H-J Lenz. The r a*-tree: an improved r*-tree with materialized data for supporting range queries on olap-data. In *Proceedings. Ninth International Workshop on Database and Expert Systems Applications*, pages 186–191, 1998.
- [JLOZ06] Christian S Jensen, Dan Lin, Beng Chin Ooi, and Rui Zhang. Effective density queries on continuouslymoving objects. In *International Conference on Data Engineering*, pages 71–71. IEEE, 2006.
- [KKT03a] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.
- [KKT03b] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, 2003.

REFERENCES

- [KKY13] Jinha Kim, Seung Keol Kim, and Hwanjo Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks? In *ICDE*, 2013.
- [KLC06] Yufeng Kou, Chang-Tien Lu, and Dechang Chen. Spatial weighted outlier detection. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 614–618. SIAM, 2006.
- [Kle03] Jon Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.
- [KM00] Flip Korn and S Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *International Conference on Management of Data*, volume 29, pages 201–212, 2000.
- [LCK03] C-T Lu, Dechang Chen, and Yufeng Kou. Algorithms for spatial outlier detection. In *International Conference on Data Mining*, pages 597–600. IEEE, 2003.
- [LCXZ12] Bo Liu, Gao Cong, Dong Xu, and Yifeng Zeng. Time constrained influence maximization in social networks. In *ICDM*, pages 439–448, 2012.
- [LCZ⁺14] Bo Liu, Gao Cong, Yifeng Zeng, Dong Xu, and Yeow Meng Chee. Influence spreading path and its application to the time constrained social influence maximization problem and beyond. *Transactions on Knowledge and Data Engineering*, 26(8):1904–1917, 2014.
- [LHT⁺12] Xingjie Liu, Qi He, Yuanyuan Tian, Wang-Chien Lee, John McPherson, and Jiawei Han. Event-based social networks: linking the online and offline social worlds. In *SIGKDD*, pages 1032–1040, 2012.
- [LKG⁺07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429, 2007.
- [LLT09] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *SIGKDD*, pages 467–476, 2009.

REFERENCES

- [LM01] Iosif Lazaridis and Sharad Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD Record*, volume 30, pages 401–412, 2001.
- [LNMG09] Yan Liu, Alexandru Niculescu-Mizil, and Wojciech Gryc. Topic-link lda: joint models of topic and author community. In *proceedings of the 26th annual international conference on machine learning*, pages 665–672. ACM, 2009.
- [LVGT12] Theodoros Lappas, Marcos R Vieira, Dimitrios Gunopulos, and Vassilis J Tsotras. On the spatiotemporal burstiness of terms. *Proceedings of the VLDB Endowment*, 5(9):836–847, 2012.
- [LWWF13] Guoliang Li, Yang Wang, Ting Wang, and Jianhua Feng. Location-aware publish/subscribe. In *Proceedings of the SIGKDD*. ACM, 2013.
- [LYM14] Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2453–2465, 2014.
- [LYS11] Junling Liu, Ge Yu, and Huanliang Sun. Subject-oriented top-k hot region queries in spatial dataset. In *Proceedings of the 20th Conference on Information and Knowledge Management*, pages 2409–2412, 2011.
- [MBK10] Michael Mathioudakis, Nilesch Bansal, and Nick Koudas. Identifying, attributing and describing spatial bursts. *Proceedings of the VLDB Endowment*, 3(1-2):1091–1102, 2010.
- [MDN12] Anirban Majumder, Samik Datta, and K.V.M. Naidu. Capacitated team formation problem on social networks. In *SIGKDD*, pages 1005–1013, 2012.
- [NAXC08] Ramesh M Nallapati, Amr Ahmed, Eric P Xing, and William W Cohen. Joint latent topic models for text and citations. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 542–550. ACM, 2008.
- [NB95] Subhas C Nandy and Bhargab B Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29:45–61, 1995.

REFERENCES

- [NG04] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.
- [NR07] Jinfeng Ni and Chinva V Ravishankar. Pointwise-dense region queries in spatio-temporal databases. In *International Conference on Data Engineering*, pages 1066–1075. IEEE, 2007.
- [OMM⁺02] Liadan O’callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeesh Motwani. Streaming-data algorithms for high-quality clustering. In *International Conference on Data Engineering*, 2002.
- [PKZT01] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *Advances in spatial and temporal databases*, pages 443–459. Springer, 2001.
- [RFP13] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. Efficient community detection in large networks using content and links. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1089–1098. ACM, 2013.
- [RWR13] Razaq Raj, Paul Walters, and Tahir Rashid. *Events Management: Principles and Practice*. SAGE Publications Ltd, 2 edition, February 2013.
- [SCFS12] Mrinmaya Sachan, Danish Contractor, Tanveer A Faruque, and L Venkata Subramaniam. Using content and interactions for discovering communities in social networks. In *Proceedings of the 21st international conference on World Wide Web*, pages 331–340. ACM, 2012.
- [SG10] Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 939–948. ACM, 2010.
- [SHK⁺10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, volume 6044, pages 456–472. Springer, 2010.

REFERENCES

- [ST11] Cheng Sheng and Yufei Tao. New results on two-dimensional orthogonal range aggregation in external memory. In *Proceedings of the thirtieth SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 129–139, 2011.
- [TBG⁺13] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 104–112. ACM, 2013.
- [THCC13] Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. Approximate maxrs in spatial databases. *Proceedings of the VLDB Endowment*, 6(13):1546–1557, 2013.
- [TK97] David Tilman and Peter M Kareiva. *Spatial ecology: the role of space in population dynamics and interspecific interactions*, volume 30. Princeton University Press, 1997.
- [TXS14] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *International Conference on Management of Data, SIGMOD*, pages 75–86, 2014.
- [WÖY⁺09] Raymond Chi-Wing Wong, M Tamer Özsu, Philip S Yu, Ada Wai-Chee Fu, and Lian Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proceedings of the VLDB Endowment*, 2(1):1126–1137, 2009.
- [WZHS07] Xuanhui Wang, ChengXiang Zhai, Xiao Hu, and Richard Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *Proceedings of the 13th ACM SIGKDD*, pages 784–793. ACM, 2007.
- [WZZ⁺14] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. Selectivity estimation on streaming spatio-textual data using local correlations. *Proceedings of the VLDB Endowment*, 8(2):101–112, 2014.
- [WZZ⁺15] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1107–1118. IEEE, 2015.

REFERENCES

- [WZZ⁺16] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Zengfeng Huang. Skype: top-k spatial-keyword publish/subscribe over sliding window. *Proceedings of the VLDB Endowment*, 9(7):588–599, 2016.
- [XYL11] Xiaokui Xiao, Bin Yao, and Feifei Li. Optimal location queries in road network databases. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, pages 804–815, 2011.
- [XZKD05] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On computing top-t most influential spatial sites. In *Proceedings of the VLDB Endowment*, pages 946–957, 2005.
- [YWN11] Da Yan, Raymond Chi-Wing Wong, and Wilfred Ng. Efficient methods for finding influential locations with adaptive grids. In *Proceedings of the 20th Conference on Information and Knowledge Management*, pages 1475–1484, 2011.
- [ZCY09] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment*, 2(1):718–729, 2009.
- [ZDXT06] Donghui Zhang, Yang Du, Tian Xia, and Yufei Tao. Progressive computation of the min-dist optimal-location query. In *Proceedings of the VLDB Endowment*, pages 643–654, 2006.
- [ZHX⁺17] Qijun Zhu, Haibo Hu, Cheng Xu, Jianliang Xu, and Wang-Chien Lee. Geo-social group queries with minimum acquaintance constraints. *The VLDB Journal*, 26(5):709–727, 2017.
- [ZLK03] Jiang Zhao, Chang-Tien Lu, and Yufeng Kou. Detecting region outliers in meteorological data. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 49–55. ACM, 2003.
- [ZS03] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the ninth ACM SIGKDD*. ACM, 2003.

REFERENCES

- [ZWL⁺11] Zenan Zhou, Wei Wu, Xiaohui Li, Mong Li Lee, and Wynne Hsu. Maxfirst for maxbrknn. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, pages 828–839, 2011.
- [ZZY⁺16] Chao Zhang, Guangyu Zhou, Quan Yuan, Honglei Zhuang, Yu Zheng, Lance M. Kaplan, Shaowen Wang, and Jiawei Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *Proceedings of the International conference on Research and Development in Information Retrieval*, pages 513–522, 2016.
- [ZZZL12] Liming Zhan, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Finding top k most influential spatial facilities over uncertain objects. In *IEEE Transactions on Knowledge and Data Engineering*, pages 922–931, 2012.