Towards Best Region Search for Data Exploration

Kaiyu Feng[1,2] Gao Cong[2] Sourav S. Bhowmick[2] Wen-Chih Peng[3] Chunyan Miao[1]
[1]LILY, Interdisciplinary Graduate School. Nanyang Technological University, Singapore
[2]School of Computer Engineering, Nanyang Technological University, Singapore
[3]Department of Computer Science, National Chiao Tung University, Taiwan
{kfeng002@e., gaocong@, assourav@, ascymiao@}ntu.edu.sg, wcpeng@cs.nctu.edu.tw

ABSTRACT

The increasing popularity and growth of mobile devices and location-based services enable us to utilize large-scale geo-tagged data to support novel location-based applications. This paper introduces a novel problem called the *best region search* (*BRS*) problem and provides efficient solutions to it. 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. This problem is fundamental to support several real-world applications such as *most influential region* search (*e.g.,* the best location for a signage to attract most audience) and *most diversified region* search (*e.g.,* region with most diverse facilities). We propose an efficient algorithm called *SliceBRS* to find the exact answer to the *BRS* problem. Furthermore, we propose an approximate solution called *CoverBRS* and prove that the answer found by it is bounded by a constant. Our experimental study with real-world datasets and applications demonstrates the effectiveness and superiority of our proposed algorithms.

1. INTRODUCTION

Due to the prominence of mobile devices and increasing popularity of location-based services (*e.g., Foursquare* (www.foursquare.com), *Yelp* (www.yelp.com)), massive amount of geo-tagged data are being generated everyday. Nowadays, in such location-based services, users can share their geo-positions with their friends. Meanwhile, *points of interests* (POIs) are increasingly tagged with category information and textual descriptions. The availability of such large-scale geo-tagged data can facilitate understanding of users who are active in a particular region or information related to the POIs in a specific region. More importantly, such data can be leveraged to search for "best" location or region based on *certain* criteria. Consider the following motivating examples of such region search problems.

Example 1: [Most Influential Region Search] Suppose a company wishes to build a signage to market their new product. The aspiration is that when people see the advertisement in the signage, they may purchase the advertised product. More importantly, these people may also recommend the product to their friends with a certain probability. The company wants to find the *best* location for the signage to attract as many people as possible to adopt the product. How can it select such location? □

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2882903.2882960

Example 2: [Most Diversified Region Search] 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 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? □

Observe that in the aforementioned scenarios, it is important for a user to specify a query rectangle of size $a \times b$ that she is interested in. For instance, in Example 2, different users may prefer regions of different sizes to explore. Hence, it is desirable for a user to specify as input the query rectangle of size $a \times b$ that she is comfortable to explore. Similarly, in Example 1, the region a company wishes to advertise its product largely depends on its business goal, interest, and budget among others. Consequently, in Examples 1 and 2, there is one running theme throughout the problems encountered, despite the differences in domain: we wish to find a rectangular region with a given size from a 2-dimensional space such that the *aggregate score* of the spatial objects in the region is maximized. Specifically, in Example 1, given the query rectangle of size $a \times b$, we wish to find the *most influential region* of the given size, such that the expected number of influenced users is maximized. On the other hand, in Example 2, given the query rectangle of size $a \times b$, we wish to find the *most diversified region* of the given size, such that the number of different categories of services and attractions in the region is maximized.

In this paper we refer to the above problem as the *best region search* (*BRS*) problem. In the aforementioned problems, the two aggregate score functions are both submodular monotone set functions as the aggregate score has a "diminishing return" property[1]. Hence, given a set of spatial objects O, a query rectangle of size $a \times b$, and a submodular monotone aggregate score function f, the *BRS* problem aims to identify an $a \times b$ rectangular region r such that the aggregate score $f(O_r)$ of all spatial objects inside the region is maximized, where O_r is the set of spatial objects inside the region r. We use an example to illustrate our problem.

Example 3: Reconsider the motivating example in Example 2. Assume that each object $o \in O$ is associated with a label to in-

[1]The marginal gain from adding an element to an input set decreases as the size of the input set increases, *i.e.,* $f(S \cup v) - f(S) \geq f(T \cup v) - f(T)$ for all elements v and all pair of sets $S \subseteq T$.

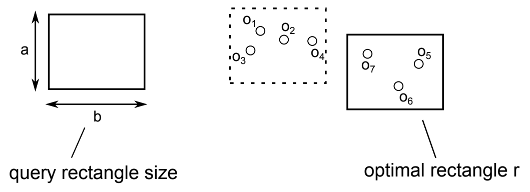


Figure 1: Example for the *BRS* problem.

dicate its category, denoted by $L(o)$. Consider the seven spatial objects $o_1, o_2, \ldots$, and o_7 shown in Figure 1 and their associated labels $L(o_1) = \ldots = L(o_5) = \{\text{Restaurant}\}$, $L(o_6) = \{\text{Mall}\}$ and $L(o_7) = \{\text{Cinema}\}$. The *BRS* problem aims to identify a most diversified region of a given size, where the diversity is measured by the number of different categories in the region. Though there are four spatial objects in the dashed-line rectangular region, all of them are restaurants and the diversity of the region is one. In the example, the solid-line rectangular region has the maximum diversity, since the spatial objects in it have three different categories. Thus, the solid-line rectangular region is an answer to the *BRS* problem. □

In the *BRS* problem, there are infinite 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 [17]. 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 [7,12,21]. Given a set O of weighted spatial points and a rectangle r of a given size, the *MaxRS* problem aims to identify a location of r such that the sum of the weights of all the points covered by r is maximized. However, this is not the case. Consider the example shown in Figure 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 (detailed in Section 2).

In this paper, we propose two novel algorithms to tackle the *BRS* problem. The first algorithm, *SliceBRS*, 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

Table 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
$\mathfrak{r}$	A disjoint region
S, s	A set of maximal slabs, a maximal slab

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.

In summary, the key contributions of this paper are as follows.

- To the best of our knowledge, this is the first work to formulate the *best region search* (*BRS*) problem for finding a rectangular region with a given size such that the submodular monotone *aggregate score* of the spatial objects inside the region is maximized.
- We develop an exact algorithm called *SliceBRS* and a constant-bounded approximate algorithm called *CoverBRS* to address the *BRS* problem.
- By applying the proposed algorithms to several real-world and synthetic datasets, we experimentally demonstrate their efficiency and effectiveness in finding the best regions in the context of most influential region search and most diversified region search problems. Furthermore, we also demonstrate how the *SliceBRS* algorithm can be adapted to address the *MaxRS* problem.

The remainder of this paper is organized as follows. The related work is reviewed in the next section. Section 3 formally introduces the best region search problem. In Section 4, we present the exact algorithm, namely *SliceBRS*. Section 5 presents the approximate algorithm *CoverBRS*. The experimental study is discussed in Section 6. Finally, the last section concludes this paper. The key notations used in this paper are given in Table 1.

2. RELATED WORK

The best region search (*BRS*) problem is related to *range aggregate query processing*, *location selection* problem, and the *region search* problem. We relate research in these areas in turn.

Range Aggregate Query Given a set O of spatial objects and a query range q, the *range aggregate* (RA) query [6, 14, 18, 22, 23] returns the total weight of the spatial objects that are inside the given query range. An RA query aim to answer "what is the total weight of objects in a given query range?" In contrast, our *BRS* problem focuses on providing answer to "where is the range of a given size such that the aggregate of objects in the range is maximized?" Clearly, these two problems are different as the queries are different. In fact, there is no systematic way to apply any method designed for RA query processing to solve the *BRS* problem. Moreover, the aggregate function used in these studies is different from the submodular monotone function used in the *BRS* problem.

Location Selection Problem Our *BRS* problem is related to the *location selection* problem.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* [31], 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 [5, 28] with road network distance. Second, given a set of servers and a set of clients, several efforts [9, 26] aim to find a location where a new server can maximize its influence. Zhou *et al.* [32] and Yan *et al.* [29] consider the problem with extensions to the definition of influence respectively. In addition, the problem of selecting top-k locations is also studied in the literature [11, 27, 30].

All the aforementioned types of location selection queries are fundamentally different from our *BRS* problem. Specifically, our *BRS* problem aims to find a region with the maximum aggregation value based on a submodular monotone function, which is different from the criteria used in location selection queries.

Region Search Liu *et al.* [20] study the problem of finding subject oriented top-k *hot regions* in spatial databases. Cao *et al.* [2] study the problem of finding a region with relevant objects from a road network, where the region is defined by a connected subgraph in the road network. The concept of region in these work is different from that of in the *BRS* problem.

Most germane to our work is the *maximizing range sum* (*MaxRS*) problem [7, 12, 21]. Given a set of weighted spatial objects, and a query rectangle, the *MaxRS* problem aims to find the location of a rectangle such that the sum of the weights of all spatial objects covered by the rectangle is maximized. The problem was first studied by the computational geometry community. Imai *et al.* propose an $O(n \log n)$ optimal algorithm [12] 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 [21] propose another line-sweeping-based algorithm with the same $O(n \log n)$ cost. The *MaxRS* problem is systematically investigated by Choi *et al.* [7] where an elegant external memory algorithm based on the in-memory $O(n \log n)$ algorithm [21] is proposed. Recently, Tao *et al.* [25] extend the *MaxRS* problem to $(1-\epsilon)$-approximate *MaxRS* problem and propose efficient algorithms for answering the *MaxRS* query approximately.

The aforementioned types of region search problem are different from the *BRS* problem as they have a different definition of region. Although the *MaxRS* problem is closest to our work, it is still a different problem from our *BRS* problem. Recall that they return different results for the example in Figure 1. We shall further validate this in Section 6. The *MaxRS* problem takes SUM as the aggregate score function, while the *BRS* problem use a general submodular monotone function as the aggregate score function. In other words, the *MaxRS* problem is a special case of our *BRS* problem when the aggregate function is SUM. In addition, techniques [7, 12, 21] developed for the *MaxRS* problem cannot be deployed or adapted to solve the *BRS* problem because their pruning techniques are tightly integrated to the SUM function and hence cannot be generalized to other submodular monotone functions.

Alexander *et al.* propose Semantic Window [15] and Searchlight [16] to study the region search problem in an interactive data exploration manner for multidimensional data. In this setting, a user explores a data space by posing a number of queries to find rectangular regions that she is interested in. In contrast to our solution to the *BRS* problem, Semantic Window [15] is build on top of PostgreSQL to search the underlying data space quickly while providing online results. Searchlight [16] combines Constraint Programming machinery and DBMS to support generic search, exploration and mining over large multi-dimensional data collections.

3. BEST REGION SEARCH (BRS) PROBLEM

In this section, we formally introduce the *best region search* (*BRS*) problem that we address in this paper. 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 [8]*]*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 $O_i \subseteq O_j \subseteq O$ 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) $f(O_i) \leq f(O_j)$.*

Example 4: 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. □

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

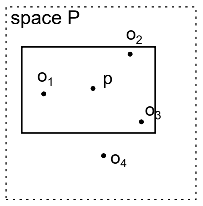
Definition 2. [**BRS** ***problem***] *Given a set of spatial objects O, a submodular monotone function $f : 2^O \rightarrow \mathbb{R}$, and the size $a \times b$ of query rectangle, the **best region search (BRS) problem** is to find a location p from the entire space P such that the aggregate score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

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

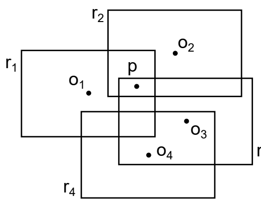
Without loss of generality, objects on the boundary of rectangles are excluded. For simplicity, we assume that no two objects have the same x coordinate or y coordinate. However, our proposed techniques can be easily extended to the case when some objects have the same x coordinate or y coordinate (detailed in Appendix A).

4. 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,



(a) The BRS problem



(b) The SIRI problem

Figure 2: Reduction from *BRS* to *SIRI*.

we first introduce the ***s**ubmodular w**ei**ghted **r**ectangle **i**ntersection* (*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). The proofs of the theorems and the lemmas of this section can be found in Appendix B.

4.1 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 $h : 2^R \to \mathbb{R}$ 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:*

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

The *BRS* problem can be reduced to the *SIRI* 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 *SIRI* problem. We illustrate the reduction with an example. Consider the example in Figure 2. In Fig 2(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 *SIRI* 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 2(b).

To prove the answer to the *SIRI* 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.*

As shown in Figure 2, 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* SIRI *problem which is reduced from the* BRS *problem, an answer to the* SIRI *problem is an answer to the* BRS *problem.*

Note that the reduction is inspired by the idea of transforming the *MaxRS* problem to the rectangle intersection problem [7, 21]. The rectangle intersection problem is defined as "Given a set of weighted rectangles, find 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.

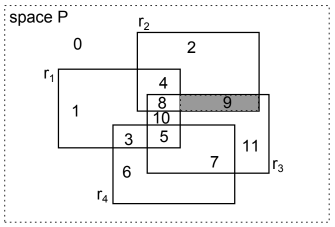


Figure 3: Disjoint regions.

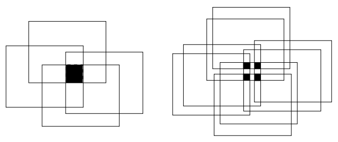


Figure 4: $O(n^2)$ maximal regions

4.2 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 $\mathfrak{r}_i$ is a **disjoint region** iff: (1) $\mathfrak{r}_i$ is the intersection of a set of rectangles R_i; (2) there exists no region $\mathfrak{r}_j$ such that (a) $\mathfrak{r}_j$ is the intersection of a set of rectangles R_j, (b) $R_i \subseteq R_j$, and (c) $\mathfrak{r}_j \subseteq \mathfrak{r}_i$. We denote the set of rectangles whose intersection is $\mathfrak{r}_i$ as $\mathfrak{r}_i.R_i$.*

Example 5: Consider the example in Fig 3. 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 $\mathfrak{r}_9$ (filled with grey color) is the intersection of the set of rectangles $\{r_2, r_3\}$. □

LEMMA 2. *All the points in a disjoint region can affect the same set of rectangles, whose intersection forms the disjoint region.*

THEOREM 2. *There are at most $O(n^2)$ disjoint regions, where n is the number of rectangles in R.*

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 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.

4.3 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 $\mathfrak{r}_i$ and the set of rectangles, denoted by $\mathfrak{r}_i.R_i$, that are affected by points in region $\mathfrak{r}_i$, we compute its weight, denoted by $h(\mathfrak{r}_i.R_i)$. 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 6: Consider the example in Fig 3. The points in disjoint region $\mathfrak{r}_3$ can affect $\{r_1, r_4\}$ and the points in disjoint region $\mathfrak{r}_5$ can affect $\{r_1, r_3, r_4\}$. The rectangles affected by the points in $\mathfrak{r}_3$

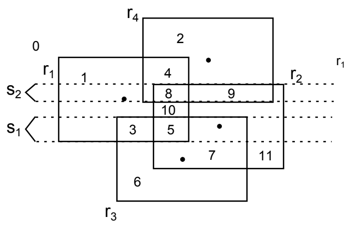


Figure 5: Example of maximal slabs.

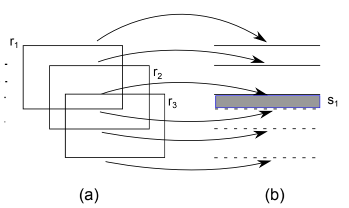


Figure 6: Find maximal slabs.

is a subset of the rectangles affected by the points in $\mathfrak{r}_5$. 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 $\mathfrak{r}_3$. □

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 $\mathfrak{r} \in \mathfrak{R}$ is defined as a* **maximal region** *if:*

(1) $\mathfrak{r}$ is a rectangular region;

(2) $\mathfrak{r}$'s left edge is part of the left edge of a rectangle in R;

(3) $\mathfrak{r}$'s right edge is part of the right edge of a rectangle in R;

(4) $\mathfrak{r}$'s top edge is part of the top edge of a rectangle in R;

(5) $\mathfrak{r}$'s bottom edge is part of the bottom edge of a rectangle in R.

Example 7: In Fig 3, $\mathfrak{r}_5$ 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, $\mathfrak{r}_8$ is also a maximal region. However, $\mathfrak{r}_3$ 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, $\mathfrak{r}_1$ is not a maximal region because it is not a rectangular region. □

LEMMA 3. *For any disjoint region $\mathfrak{r}_i$, there exists a maximal region $\mathfrak{r}_j$ such that $A(p_{\mathfrak{r}_i}) \subseteq A(p_{\mathfrak{r}_j})$, where $p_{\mathfrak{r}_i}$ is a point from $\mathfrak{r}_i$ and $p_{\mathfrak{r}_j}$ is a point from $\mathfrak{r}_j$.*

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.*

4.4 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.*

Example 8: We use the example in Figure 6 to illustrate maximal slab. For each rectangle, we extend its top edge and bottom edge to get two horizontal lines, as shown in Figure 6(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 6 (b) is the only maximal slab. □

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

LEMMA 5. *For any maximal region $\mathfrak{r}$, there exists at least one maximal slab s such that $\mathfrak{r}$ intersects with s.*

LEMMA 6. *There are at most $O(n)$ maximal slabs, where n is the number of rectangles.*

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 $upper(s)$.*

Example 9: Consider the maximal slab s_2 in Fig 5. We compute $upper(s_2) = h(\{r_1, r_2, r_4\})$. For any point p in s_2, we have $h(A(p)) \leq upper(s_2)$. □

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

Function ScanSlab(R)

```
Output: A list of maximal slabs $S$.
1  $Rec \leftarrow \emptyset, S \leftarrow \emptyset, flag \leftarrow$ 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          $flag \leftarrow$ bottom;
6      else
           /* The sweep line meets the top edge
              of a rectangle                      */
7          if $flag =$ bottom then
8              $s \leftarrow$new maximal slab, $s.UB = h(Rec)$;
9              $S \leftarrow S \cup \{s\}$;
10         $Rec \leftarrow Rec \setminus \{r\}$;
11         $flag \leftarrow$ top;
12 return $S$;
```

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                 $p \leftarrow$ 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 $p_c$;
```

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.

Example 10: Suppose we invoke the `ScanSlab` procedure on the example shown in Figure 5(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 $Rec = [r_3, r_2, r_1]$, $flag =$ bottom. After that, the sweep line comes across the top edge of r_3. Since $flag =$ 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 $Rec = [r_2, r_1, r_4]$, $flag =$ bottom. Another maximal slab s_2 is found. We assume that $h(\{r_2, r_1, r_4\}) = 4$ and s_2 is pushed into S. The algorithm terminates when the sweep line meets the top edge of r_4. □

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 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.

4.5 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 6).

Example 11: Consider the example in Fig 7. 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. □

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.*

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:

Algorithm 1: SliceBRS

```
Input: A set of objects $O$, query size $a \times b$, aggregate function $f$
Output: A point $p$
1  $p_c \leftarrow$ 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      $S_i \leftarrow$ 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      $p \leftarrow$ SearchMR($R_i, S_i$);
9      if $h(A(p)) > h(A(p_c))$ then
10         $p_c \leftarrow p$;
11 return $p_c$;
```

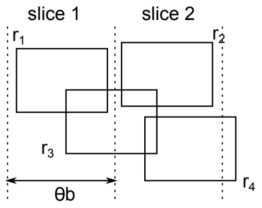


Figure 7: Equal-width slices.

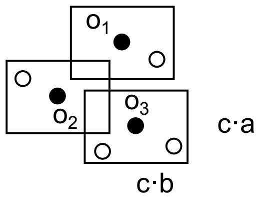


Figure 8: c-cover

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.*

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.

4.6 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* 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 analyse 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 $O(n)$.*

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).

5. 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 6). Hence in this section, we present a constant-bounded approximate algorithm called *CoverBRS* for answering the *BRS* problem efficiently especially for large datasets. The proofs of the theorems and the lemmas of this section can be found in Appendix B.

5.1 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.

5.2 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 object 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.*

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

Example 12: Consider the example shown in Fig 8. 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. □

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 5.6.

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 an $(1-c)a \times (1-c)b)$ rectangular region centered at p can cover t, then the $a \times b$ rectangular region centered at p can cover all objects in $O_{r_t^{ca,cb}}$.*

5.3 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.*

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 [3], 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 regions[2] until each leaf node contains one spatial object.

[2]We use a minimum bounding rectangle (MBR) to enclose all spatial objects, thus we refer to the partitions as "rectangular regions" instead of "quadrants".

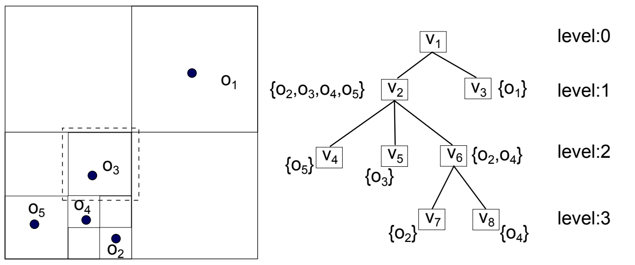


Figure 9: Selection of c-cover T.

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 13: Consider the example in Figure 9. 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. □

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., *$V_l^1 = \{v.t | v$ 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., *$V_l^2 = \{v.t | v$ 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.*

Example 14: Consider the example in Figure 9. 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. □

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:

$$l = \max\left(\lceil \log_2 \frac{Height}{ca} \rceil, \lceil \log_2 \frac{Width}{cb} \rceil\right). \qquad (1)$$

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 spatial objects O and a parameter c. We use a

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 $p$;
```

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 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.

5.4 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: $f_T : 2^T \to \mathbb{R}$, which maps a subset of T to a real number such that for any $T_i \subseteq T$ and $T_i = \{t_1, \ldots, t_j\}$*

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

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 $(1-c)a \times (1-c)$ 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.

5.5 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.

5.6 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. The proofs of the following lemma and theorems can be found in Appendix B.

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:*

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

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

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

THEOREM 6. *When $c = 1/2$, Algorithm 2 returns a (1/9)-approximate answer to the optimal solution.*

THEOREM 7. *The 1/9 approximation ratio is tight for the* Cover-BRS *algorithm when $c = 1/2$.*

6. 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 applicaitons. 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.

6.1 Experimental Setup

Real-world applications Recall from Section 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 [17]. 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 [17] 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.

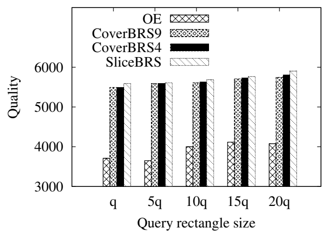
Application 2: Most Diversified Region. The second application of the *BRS* problem is to find the most diversified region. Specifically, given the size of a region, we can apply our algorithms to find a region with maximum diversity. We consider a set of spatial objects, each of which is associated 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 2: Summary of datasets for Application 1.

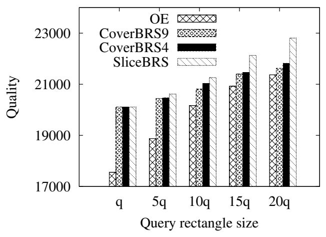
Property	Brightkite	Gowalla
# of objects	693,362	1,256,692
# of check-ins	4,491,143	6,442,890
# of users	58,228	196,591
Width	314.391	495.658
Height	359.823	353.771

Table 3: 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



(a) Brightkite



(b) Gowalla

Figure 10: Quality vs. kq.

Datasets To support evaluation of our algorithms for the first application, we use two real-life datasets[3], `Brightkite` and `Gowalla`, whose properties are given in Table 2. We use another two datasets, `Meetup` and `Yelp`[4], to evaluate our algorithms for the second application. The properties of these two datasets are reported in Table 3. In Appendix C.1, we present a more detailed description of the datasets.

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 [21] 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 Appendix C.2.

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 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 $Weight$ 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 $f(.)$ in our problem definition. In the subsequent discussion, we refer to the latter measure as "quality" of the results.

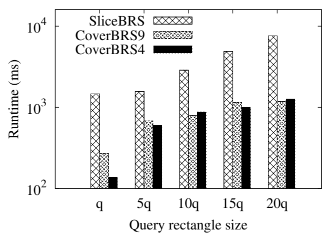
6.2 Quality and Efficiency

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 $20q$.

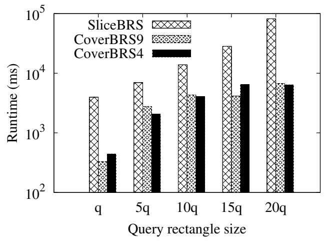
Application 1 (Most Influential Region). We first study the performance of our algorithm for Application 1, *i.e.,* finding the most influential region, in terms of both quality and runtime. Figure 10 reports the quality of the returned region for query rectangles of dif-

[3]http://snap.stanford.edu/data/index.html

[4]http://www.yelp.com.sg/dataset_challenge

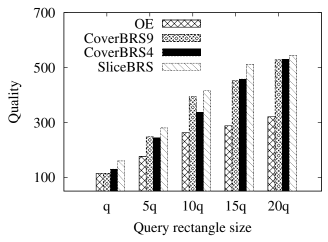


(a) Brightkite

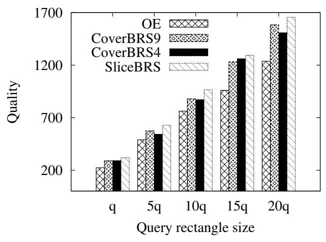


(b) Gowalla

Figure 11: Runtime vs. kq.



(a) Yelp



(b) Meetup

Figure 12: Quality vs. kq.

ferent 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 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.

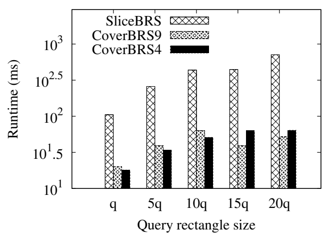
Application 2 (Most Diversified Region). Next, we investigate the performance of our algorithm for Application 2. Figure 12 and Figure 13 report the quality and runtime of the evaluated algorithms, respectively. We can make similar observations as those on Figure 10 and Figure 11.

6.3 Usefulness of Optimization Strategies

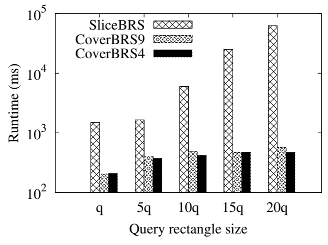
We conduct several experiments to validate the usefulness of the various strategies we proposed in Sections 4 and 5 to improve efficiency of our proposed algorithms.

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 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.

Usefulness of Maximal Slab Next, we investigate the usefulness of maximal slabs in improving the efficiency of our algorithm. By



(a) Yelp



(b) Meetup

Figure 13: Runtime vs. kq.

Table 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

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 Table5. 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. 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 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 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.

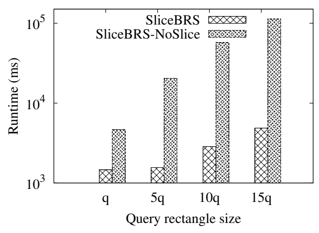
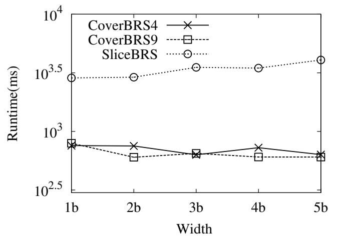
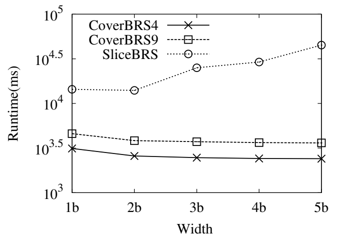


Figure 14: Runtime of *SliceBRS* with and without cutting the space into slices.



(a) Brightkite



(b) Gowalla

Figure 15: Runtime vs. slice width in Application 1.

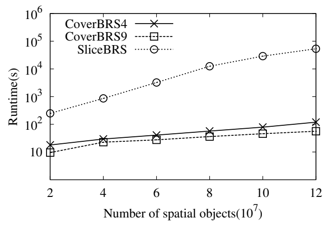


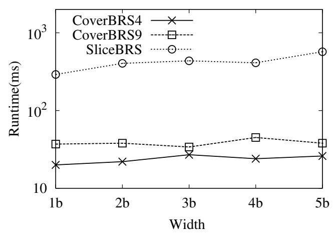
Figure 16: Runime vs. graph size.

Table 5: Effectiveness of maximal slabs.

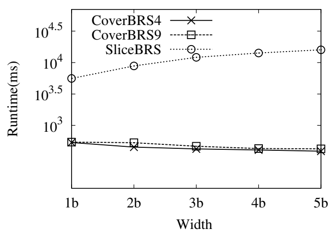
	Dataset	#MR	#MS	#MSP	#DRP
App. 1	Brightkite	247,332	649,442	39	1,249
	Gowalla	349,723	1,206,367	132	8,417
App. 2	Yelp	40,460	33,699	170	6,342
	Meetup	167,789	516,379	3,257	199,975

Table 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



(a) Yelp



(b) Meetup

Figure 17: Runtime vs. slice width in Application 2.

6.4 Effect of Parameters

Next, we investigate the effect of parameters on our proposed algorithms. Specifically, we study the effect of slice width here. Due to space constraints, the effect of the query rectangle on the performance of our algorithms is reported in Appendix C.3.

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 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 17. We can make similar observations as those on Figure 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 slices. This set of experiments proves again that the idea of cutting space into slices greatly improves the efficiency of *SliceBRS*.

6.5 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 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 [7], 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 16. 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.

7. CONCLUSIONS AND FUTURE WORK

The quest for high quality location-based services has become more pressing due to explosive growth of mobile devices and geotagged data. In this paper, 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.

Acknowledgments This research is supported by the National Research Foundation Singapore under its Interactive Digital Media (IDM) Strategic Research Programme, a Singapore MOE AcRF Tier 2 Grant (ARC30/12), and a Singapore MOE AcRF Tier 1 Grant (RG22/15). Wen-Chih Peng was supported in part by MOST Taiwan, Project No. 102-2221-E-009 -171 -MY3 and 104-2221-E-009 -138 -MY2.

8. REFERENCES

[1] C. Borgs, M. Brautbar, J. Chayes, and B. 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.

[2] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *Proceedings of the VLDB Endowment*, 7(9):733–744, 2014.

[3] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[4] W. Chen, C. Wang, and Y. 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.

[5] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long. Efficient algorithms for optimal location queries in road networks. In *International Conference on Management of Data, SIGMOD*, pages 123–134, 2014.

[6] H.-J. Cho and C.-W. Chung. Indexing range sum queries in spatio-temporal databases. *Information and Software Technology*, 49(4):324–331, 2007.

[7] D.-W. Choi, C.-W. Chung, and Y. Tao. A scalable algorithm for maximizing range sum in spatial databases. *Proceedings of the VLDB Endowment*, 5(11):1088–1099, 2012.

[8] M. Conforti and R. 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.

[9] Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *Advances in Spatial and Temporal Databases*, pages 163–180. Springer, 2005.

[10] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.

[11] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In *Proceedings of the 20th Conference on Information and Knowledge Management*, pages 2377–2380, 2011.

[12] H. Imai and T. 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.

[13] K. Jung, W. Heo, and W. Chen. Irie: Scalable and robust influence maximization in social networks. In *International Conference on Data Mining, ICDM*, pages 918–923, 2012.

[14] M. 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.

[15] A. Kalinin, U. Cetintemel, and S. Zdonik. Interactive data exploration using semantic windows. In *International Conference on Management of Data, SIGMOD*, pages 505–516, 2014.

[16] A. Kalinin, U. Cetintemel, and S. Zdonik. Searchlight: enabling integrated search and exploration over large multidimensional data. *Proceedings of the VLDB Endowment*, 8(10):1094–1105, 2015.

[17] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[18] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD Record*, volume 30, pages 401–412, 2001.

[19] B. Liu, G. Cong, Y. Zeng, D. Xu, and Y. M. 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.

[20] J. Liu, G. Yu, and H. 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.

[21] S. C. Nandy and B. B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29:45–61, 1995.

[22] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *Advances in spatial and temporal databases*, pages 443–459. Springer, 2001.

[23] C. Sheng and Y. 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.

[24] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *International Conference on Management of Data, SIGMOD*, pages 75–86, 2014.

[25] Y. Tao, X. Hu, D.-W. Choi, and C.-W. Chung. Approximate maxrs in spatial databases. *Proceedings of the VLDB Endowment*, 6(13):1546–1557, 2013.

[26] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proceedings of the VLDB Endowment*, 2(1):1126–1137, 2009.

[27] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proceedings of the VLDB Endowment*, pages 946–957, 2005.

[28] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, pages 804–815, 2011.

[29] D. Yan, R. C.-W. Wong, and W. 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.

[30] L. Zhan, Y. Zhang, W. Zhang, and X. Lin. Finding top k most influential spatial facilities over uncertain objects. In *IEEE Transactions on Knowledge and Data Engineering*, pages 922–931, 2012.

[31] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *Proceedings of the VLDB Endowment*, pages 643–654, 2006.

[32] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. Maxfirst for maxbrknn. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, pages 828–839, 2011.

APPENDIX

A. EXTENSION TO OBJECTS WITH SAME COORDINATES

In this paper, we consider the case that the coordinates of each spatial object is unique. In real-life, it is possible that several spatial objects share the same x 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.

B. PROOFS

Proof of Lemma 1:

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 $r_o^{a,b}$. $\square$

Proof of Theorem 1:

PROOF. Let P be the space in the *BRS* problem and the *SIRI* 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. $\square$

Proof of Lemma 2:

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. $\square$

Proof of Theorem 2:

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. $\square$

Proof of Lemma 3:

PROOF. We first consider two horizontally consecutive disjoint regions, $\mathfrak{r}_1$ and $\mathfrak{r}_2$. They are separated by a vertical edge from rectangle r. There are two cases:

Case 1: The vertical edge is the left edge of rectangle r, and then the points in disjoint region $\mathfrak{r}_2$ are inside rectangle r. The points in $\mathfrak{r}_2$ can affect rectangle r besides all rectangles that the points in $\mathfrak{r}_1$ can affect. In other words, the set of rectangles affected by points in $\mathfrak{r}_1$ is a subset of the rectangles affected by points in $\mathfrak{r}_2$.

Case 2: The vertical edge is the right edge of rectangle r, then the points in disjoint region $\mathfrak{r}_1$ are inside rectangle r. We can draw a similar conclusion that the set of rectangles affected by points in $\mathfrak{r}_1$ is a subset of the rectangles affected by points in $\mathfrak{r}_2$.

For two vertically consecutive disjoint regions, we have similar conclusions.

Given a disjoint region $\mathfrak{r}_i$, if it is a maximal region, then Lemma 3 holds, where $\mathfrak{r}_i = \mathfrak{r}_j$. Otherwise, we can always find a neighboring disjoint region $\mathfrak{r}_j$ of $\mathfrak{r}_i$ such that $A(p_{\mathfrak{r}_i}) \subseteq A(p_{\mathfrak{r}_j})$. Since there are finite disjoint regions, we will find a sequence of disjoint regions $< \mathfrak{r}_i, ..., \mathfrak{r}_k >$ and $\mathfrak{r}_k$ is a maximal region such that $A(p_{\mathfrak{r}_i}) \subseteq ... \subseteq A(p_{\mathfrak{r}_k})$.

Put these together, we have the conclusion. $\square$

Proof of Lemma 4:

PROOF. We use an example to show that n rectangles can generate $O(n^2)$ maximal regions. As shown in Fig 4, the black regions are maximal regions. When there are 4 rectangles, there is $1 \times 1 = 1$ maximal region. When there are 8 rectangles, there are $2 \times 2 = 4$ maximal regions. Similarly, when there are n rectangles, there are $n/4$ maximal regions each row and there are $n/4$ rows. So there could be $n^2/16 = O(n^2)$ maximal regions in the worst case. $\square$

Proof of Lemma 5:

PROOF. For a maximal region $\mathfrak{r}$, 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. $\square$

Proof of Lemma 6:

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. $\square$

Proof of Lemma 7:

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. $\square$

Proof of Lemma 8:

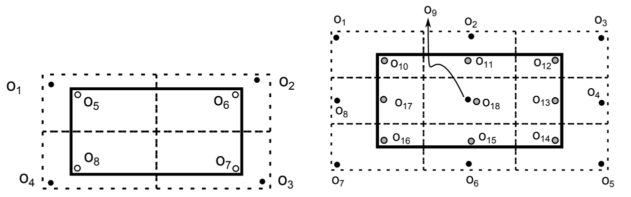
PROOF. We project each rectangle to the x-axis and get a line segment with a length of b. Similarly, we project each slice to the x-axis and get a line segment with a length of θb. A rectangle intersects with a slice iff their projections on x-axis intersect. The projection of a rectangle can intersect 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. $\square$

Proof of Lemma 9:

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. $\square$

Proof of Lemma 10:

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 analyse 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



(a) $c = 1/3$. (b) $c = 1/2$.

Figure 18: Approximate Ratio.

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 $O(\frac{\theta b \cdot \sqrt{n}}{W} \cdot \sqrt{n} \cdot \frac{W}{b}) = O(n)$ time to find the best point. Putting these together, we can conclude that the complexity of *SliceBRS* for this case is $O(n)$. □

Proof of Lemma 11:

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. □

Proof of Theorem 3:

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 [10]. 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. □

Proof of Lemma 12:

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 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. □

Proof of Lemma 13:

PROOF. We have

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

□

Proof of Theorem 4:

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 $r_1, \ldots, r_4$ such that for any object o covered by $r_{p_{opt}}^{a,b}$, its representor is covered by $r_1 \cup \ldots \cup r_4$. Consider the Example in Figure 18a, objects $o_1, \ldots, o_4$ represent $o_5, \ldots, o_8$ respectively and $o_1, \ldots, o_4$ are covered by $r_1 \cup \ldots \cup r_4$. Thus, we can achieve the following derivation:

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

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

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

□

Proof of Theorem 5:

PROOF. We prove this with a worst case example. Consider an instance in Figure 18a 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 $f(\{o_1\}) = \ldots = f(\{o_4\}) = 1$, $f(\{o_5\}) = \ldots = f(\{o_8\}) = 1 - \beta$, where β is a small positive real number, $f(\{o_1, o_5\}) = \ldots = f(\{o_4, o_8\}) = 1$, and $f(\{o_4, \ldots, o_8\} = 4 - 4\beta$. In this case, the *CoverBRS* algorithm may choose any of $o_1, \ldots, o_4$ 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. □

Proof of Theorem 6:

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 $r_1, \ldots, r_9$ to guarantee that for any object $o \in O$ covered by $r_{p_{opt}}^{a,b}$, its representor is covered by $r_1 \cup \ldots \cup r_9$, as illustrated in Figure 18b. Thus, we can achieve the following derivation:

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

From Lemma 13, we have

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

□

Proof of Theorem 7:

PROOF. We prove this by giving a worst case example. Consider an instance in Figure 18b 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 $T = \{o_1, \ldots, o_9\}$ and they represent $o_{10}, \ldots, o_{18}$ respectively. We assume $f(\{o_1\}) = \ldots = f(\{o_8\}) = 1$ and $f(\{o_9\}) = \ldots = f(\{o_{17}\}) = 1 - \beta$ and $f(\{o_{18}\}) = \beta$ where β is a small positive real number. We assume $f(\{o_1, o_{10}\}) = \ldots = f(\{o_8, o_{17}\}) = 1$ and $f(\{o_9, o_{18}\}) = 1 - \beta$. We also assume that $f(\{o_{10}, \ldots, o_{18}\} = 9 - 9\beta$. In this case, since the *CoverBRS* algorithm only consider the objects in the c-cover which locate at $o_1, \ldots, o_8$, we may end up choosing any one of $o_1, \ldots, o_8$

as the approximate solution. However, the optimal solution should be the center of the solid-line rectangle and it aggregation is $9 - 9\beta$. Since β can be any small positive value, our answer is (1/9)-approximate. $\square$

C. DATASET DESCRIPTION AND MORE EXPERIMENTAL RESULTS

C.1 Datasets Description

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 [4, 13, 19], *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 [1, 24], 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.

C.2 Application to the MaxRS Problem

Recall that the *MaxRS* problem is a special case of the *BRS* problem as discussed in Section 2. 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 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.

Table 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%

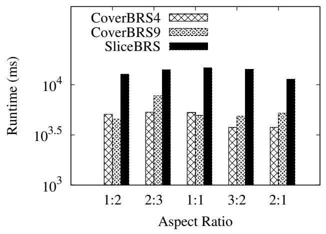


Figure 19: Effect of Aspect Ratio.

C.3 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 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 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 12 reports the quality of the returned region when we vary the size of query rectangles in the Application 2. Figure 13 presents the runtime of the algorithms. We can make similar observations as those on Figures 10 and 11.

Lastly, we study the effect of the aspect ratio of the query rectangle. We use 5 aspect ratios, 1:2, 2:3, 1:1, 3:2, 2:1, i.e., the query rectangle varies from wide rectangle to tall rectangle. Due to the limitation of space, we only report the results on `Gowalla`. Similar observations can be made on other datasets. Figure 19 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.