



DARS: Diversity and Distribution-Aware Region Search

Siyu Liu¹, Qizhi Liu¹(✉), and Zhifeng Bao²

¹ State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

mg1833049@smail.nju.edu.cn, lqz@nju.edu.cn

² RMIT University, Melbourne, Australia
zhifeng.bao@rmit.edu.au

Abstract. Recent years have seen the rapid development of Location Based Services (LBSs). Many users of these services are making use of them to, for example, plan trips, find houses or explore their surroundings. In this paper we introduce a novel problem called the *diversity and distribution-aware region search* (DARS) problem. In particular, DARS aims to find regions of size $a \times b$ where the number of different categories is maximized such that objects of different categories are not too scattered from each other and objects of the same category are within reasonable distance (which is a tunable parameter to cater for different users' needs). We propose several methods to tackle the problem. We first design a sweepline based method, and then design various techniques to further improve the efficiency. We have conducted extensive experiments over real datasets and demonstrate both the usefulness and the efficiency of our methods.

Keywords: Spatial region search · Spatial databases · Points of interest

1 Introduction

With the quick growth of Location Based Services (LBSs) in recent years, many online businesses (e.g., *Yelp*¹, *Dianping*²) are now incorporating geographic data such as point of interest (POI) in their services. Apart from geographic locations, POIs are also often tagged with category information. Those data can be utilized to help explore a large area more efficiently, such as finding a best location for facility deployment [4–6, 18–20]. Apart from finding a single best location, the study [7] by Feng et al. aimed to identify a “best region” that contains the largest number of POI categories. In our work we argue that interrelationship among the POIs within a region is also important, as illustrated in the following example.

¹ www.yelp.com.

² www.dianping.com.

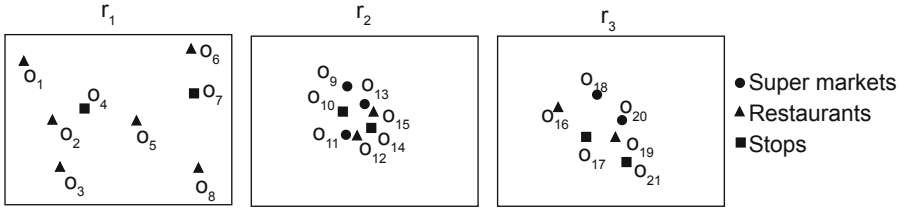


Fig. 1. A motivating example.

Example 1. John Doe plans to buy a new property in Anytown, and he wants to first find a region that has many kinds of facilities (e.g., POIs such as super markets, restaurants, metro or bus stops, parking lots etc.). If the locations of such POIs in a region are not too scattered, then John can make use of them quite efficiently without having to waste time on commuting. John might also prefer that there are at least some candidates for each category so he would have backup options if some POIs of certain category cannot satisfy his needs (e.g., different operation hours). How can John select such a “diversified” and “well-distributed” region in Anytown so that he can choose the location of his new property regarding the region?

Figure 1 shows three possible candidate regions for John. Although r_1 contains eight POIs, which is more than that in r_2 and r_3 , r_1 contains POIs of two categories while r_2 and r_3 contain three, thus on diversity r_1 loses. Furthermore, POIs of different categories in r_1 are much scattered than those in r_2 and r_3 , hence in r_2 or r_3 John can spend less time and energy traveling to different POIs. In this case r_2 and r_3 shall be considered as answers prior to r_1 . We can even say r_2 is better than r_3 because POIs of different categories are even closer in r_2 . For POIs of one category (e.g., restaurants), it is better to have some POIs of the same category nearby so John can have more options. Regarding this we can see that r_2 is also better than r_3 because POIs of same categories are also closer in r_2 . These distribution of POIs within a region is the interrelationship of POIs we want to consider. At the same time, there might be other regions just as good as r_2 . Therefore, returning multiple regions helps when there are multiple relatively good regions. As a result, with less time spent on commuting between different POIs, John can quickly find public transportations during his work in the daytime, have dinners of different styles to explore different dining options and shop at different markets to find the general goods he need in r_2 . Note that it is critical that John offers the size of such a region that is of interest to him. In fact, other users might also want to search for a region satisfying their needs the most, but of different size.

From the example we can see that, given the size of a query rectangle, users wish to find regions that are **diversified** and the POIs within the regions are **well-distributed**, such that the number of different categories is maximized

Table 1. Table of notations.

Notation	Description
O	The set of spatial objects
C	The set of categories
R	The set of regions
T	The set of intersections
o, r, t	An object, a region, an intersection
$a \times b$	Size of a region
O_r	The set of spatial objects covered by r
C_r	The set of categories region r possesses
O_{r_m}	All POIs of category m in region r
$o_{r_m}^i$	The i th object of category m in r
$d'(O_{r_m}, O_{r_n})$	Minimum distance between two categories

while POIs of different categories are not too scattered from each other. As for POIs of the same category, different users might have different requirements for their distribution. This kind of interrelationship between POIs in a region is never considered in region search to the best of our knowledge. We refer to the problem in our paper as *diversity and distribution-aware region search* (DARS) problem. Briefly, in a spatial object database O , given a query rectangle of size $a \times b$, DARS aims to find regions that have the most diversified collection of POIs, minimize the average distance among POIs of different categories, and POIs of the same categories hold a distance no greater than a threshold between each other. In summary, we make the following contributions (Table 1):

- We propose and study the DARS problem for the first time, to capture diversity and distribution of POIs in a region simultaneously. Specifically, we use distance to capture the interrelationship among POIs in a region to measure the distribution at a region level (Sect. 2).
- We first design a baseline algorithm for scanning and finding regions, and then propose two optimizations to further improve the efficiency (Sect. 3).
- We conduct extensive experiments over two real-world and one synthetic datasets to demonstrate the efficiency and effectiveness of our solutions (Sect. 4).

2 Problem Formulation

In this section we formally introduce the *diversity and distribution-aware region search* (DARS) problem, starting from the following two preliminaries to define the diversity.

Definition 1 (Average Minimum Distance). *The average minimum distance among all POIs of different categories is:*

$$g(O_r) = \frac{1}{|C|} \sum_{1 \leq m < n \leq |C|} d'(O_{r_m}, O_{r_n}).$$

When considering the diversity of a region, we aim to minimize the average distance among POIs of different categories. For a region r , we can get the average minimum distance with $g(O_r)$. In $g(O_r)$, we use the minimum distance between POIs of different categories. The reason is that if the average minimum distance of POIs of different categories in a region is small, not only that POIs of different categories are quite near to each other, but also that we can avoid impacts from stray points. As a result users can spend less time in a region finding POIs that satisfy their different needs. We then need to find a way to define a “score” for the diversity of a region. We have two intuitions to follow. First, regions with more categories shall be considered prior to regions with less categories. Second, when two regions have the same number of categories, the distribution of POIs will most likely differ. Intuitively, POIs of different categories should be close to each other so that users can utilize different services efficiently. At the same time, for POIs of each category there should be some POIs of the same category nearby for users to have more candidates to choose from.

Definition 2 (In-Region Diversity). *Given a region r , its in-region diversity of r is defined as*

$$f(r) = e^{-\left(\frac{|C_r| \times (|C_r| - 1)}{2} + e^{-g(O_r)}\right)}.$$

The definition of in-region diversity is driven by the two intuitions above. $\frac{|C_r| \times (|C_r| - 1)}{2}$ is always monotone to $|C_r|$, so the difference with different $|C_r|$ is always greater than one. For example, consider $|C_r| = 3$, then $\frac{|C_r| \times (|C_r| - 1)}{2} = 3$, $e^{-g(O_r)}$ is a value in range $(0, 1]$, so $\frac{|C_r| \times (|C_r| - 1)}{2} + e^{-g(O_r)}$ is a value in range $(3, 4]$. But when $|C_r| = 2$, $\frac{|C_r| \times (|C_r| - 1)}{2} + e^{-g(O_r)}$ is a value in range $(1, 2]$. These two ranges never intersect, so regions with more categories is always superior. From Fig. 1 in Example 1 we can see that r_2 and r_3 both possess three categories, and POIs of different categories are closer to each other in r_2 than those in r_3 , then $g(O_{r_2}) < g(O_{r_3})$, so $f(r_2) < f(r_3)$. Furthermore, POIs of same categories are also closer to each other in r_2 . In fact, this is just one possible form of the function f . Any other forms should also work as long as they consider regions with more categories prior to other regions, or as long as they consider regions that have POIs of different categories closer to each other prior to other regions.

To this end, we proceed to present our problem formulation as below.

Definition 3 (Diversity and distribution-aware Region Search (DARS)). *In a spatial object dataset O , given a query rectangle size $a \times b$, a diversity function f , and a distance threshold δ , DARS finds a region r of size $a \times b$ that minimizes $f(r)$, while the distance between at least one pair*

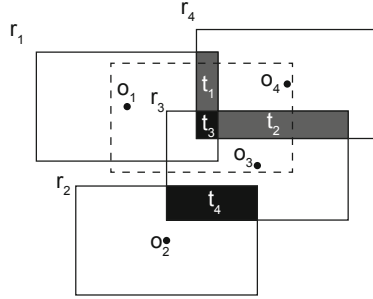


Fig. 2. Rectangle intersections.

of POIs of the same category should be no greater than a threshold δ : Formally, DARS finds a region r i.e., $r = \arg \min_{r \in R} f(r)$, subject to the constraint $\exists i, j \ d(o_{r_m}^i, o_{r_m}^j) \leq \delta, 1 \leq m \leq |C|$.

Note that the constraint requires a distance threshold δ . It means that there should be at least a pair of POIs of the same category that hold a distance of at most δ from each other. This is a tunable parameter to cater for needs in different preferences and circumstances – some users might think that the more the better, while others might just want a small but enough amount of POIs around. When δ becomes the diagonal’s length of the query rectangle, there is no constraint for POIs of the same category. We use as δ – constraint to denote this constraint set for POIs of the same category in a region. Moreover, the definition can be generalized so that we can even consider the constraint for POIs of different categories and try to minimize the minimum average distance among POIs of the same category. This is how the DARS problem can be generalized. In this paper we follow the case in Definition 3.

3 Our Solutions

In this section we explain the details of our solutions. We first show that the DARS problem can be reduced to the rectangle intersection problem. Then we propose a baseline method called DPOF to solve the rectangle intersection problem in order to find the answers to the DARS problem. However, DPOF lacks in efficiency as we will see in experiments, so we present an optimization strategy. Last, we further optimize the method via a space partitioning to enhance the efficiency.

3.1 Finding Regions

Before we score a region, we must find one. Suppose a user issues a query request with a rectangle query range of size $a \times b$, the goal is to find some candidate regions in the whole search space (e.g., search in a city or a province) as results.

Algorithm 1: FindMaxIntersects

Input: A set R of rectangles
Output: All maximal intersections T

```

1  $T \leftarrow \emptyset, G \leftarrow \emptyset;$ 
2 while Sweep a vertical line  $l_v$  from left to right do
3   if  $l_v$  meets the left of a rectangle  $r$  then
4      $G \leftarrow G \cup \{r\};$ 
5   else if  $l_v$  meets the right of a rectangle  $r$  then
6     if A left edge and a right edge has been scanned continuously then
7       Get the maximal intersection  $t$  of rectangles in  $G;$ 
8       if  $t$  is not fully covered by any intersection in  $T$  then
9         Delete intersections that are covered by  $t$  from  $T;$ 
10       $T \leftarrow T \cup \{t\};$ 
11       $G \leftarrow G \setminus \{r\};$ 
12 return  $T;$ 

```

Algorithm 2: DARS Post-Filter (DPOF)

Input: A set R of rectangles of size $a \times b$
Output: Candidate regions R_c

```

1  $R_c \leftarrow \emptyset;$ 
2  $T \leftarrow \emptyset, A \leftarrow \emptyset;$ 
3 while Sweep a horizontal line  $l_h$  from bottom to top do
4   if  $l_h$  meets the bottom of a rectangle  $r$  then
5      $A \leftarrow A \cup \{r\};$ 
6   else if  $l_h$  meets the top of a rectangle  $r$  then
7     if A bottom edge and a top edge has been scanned continuously then
8        $T' \leftarrow \text{FindMaxIntersects}(A);$ 
9        $T \leftarrow T \cup T';$ 
10       $A \leftarrow A \setminus \{r\};$ 
11 foreach  $t$  in  $T$  do
12    $O_t \leftarrow$  Set of objects at the centers of rectangles that form  $t;$ 
13   if  $O_t$  satisfies the  $\delta$ -constraint then
14      $r_c \leftarrow$  a new region of size  $a \times b$ , centered at a the center of  $t;$ 
15     //  $r_c$  contains all objects in  $o_t$ 
16      $R_c \leftarrow R_c \cup \{r_c\};$ 
17 return  $R_c$ 

```

Unfortunately, it is prohibitively expensive to search the whole space because there is an infinite number of rectangles of size $a \times b$.

Reduction of the DARS Problem. We first reduce the step of finding regions to the problem of *max-enclosure* in [14]. The goal of *max-enclosure* is to find the position of a rectangle that encloses a maximum number of points. This enclosure problem can be transformed to the *rectangle intersection* problem according

to [14]. Hence our first goal becomes finding where the most rectangles intersect in a given set of rectangles.

Consider the example in Fig. 2. There are four objects, and we draw rectangles with the same size $a \times b$ centered at each object. The highlighted areas are possible intersections. r_2 and r_3 form an intersection t_4 . r_1 and r_4 form an intersection t_1 , r_4 and r_3 form an intersection t_2 . Here, t_1 and t_2 again intersect and form an intersection t_3 . We can see that t_3 is formed by a set of three rectangles r_1 , r_3 and r_4 , which is a superset of the rectangles that form t_1 and t_2 . As a result, for these four rectangles we find two “most important” intersections t_3 and t_4 . For an intersection we find, we can simply choose an arbitrary point inside the intersection and draw a rectangle of size $a \times b$ centered at the point. The newly drawn rectangle will cover the centers of all rectangles that form the intersection. For instance, the dashed-line rectangle can cover o_1 , o_3 and o_4 because it is centered at an point inside t_3 . Hence we can find all such intersections and rate the corresponding regions to get the answers to the DARS problem.

Finding Intersections. From the example in Fig. 2 we know that we only need to find and check those “most important intersections” because they are formed by a maximal number of rectangles among all intersections. We refer to these most important intersections as *maximal intersections*. In general, we use the sweep-line method to find the maximal intersections. We summarize the steps in Algorithm 1. For a set of rectangles, we can scan from left to right (Line 1.2–1.11) and during the scanning we can check if a possible group of rectangles is a superset of other groups that have already been found. If they do include some other groups, we discard those groups and preserve this newly found group (Line 1.8–1.10)

With the help of Algorithm 1, we can start scanning all rectangles to find regions. We summarize the method in Algorithm 2. We use a horizontal line to scan from the bottom to the top. If the line meets the bottom of a rectangle, we add this rectangle to an *Active rectangles* set. If the line meets the top of a rectangle, we remove this rectangle from the *Active rectangles* set (Line 2.3–2.10). If the line meets a bottom edge and a top edge consecutively, it means that we can find some possible groups of intersecting rectangles from *Active rectangles* set now (Line 2.7–2.9). The whole scanning process terminates until the topmost horizontal edge is met. We can get a series of regions by drawing rectangles centered at a point in intersections attained from Algorithm 1. Finally we can filter out the regions that does not satisfy the δ – *constraint*, then we have the remaining regions as results (Line 2.11–2.15).

With Algorithm 1 and Algorithm 2, we now have a basic workflow of solving the DARS problem. In the following sections we will focus on the optimization of this workflow.

3.2 Checking the δ – *Constraint* During Sweeping

Algorithm 1 finds all maximal intersections. When the size of O grows or the query rectangle size grows, it is time-consuming to find all maximal intersections.

Algorithm 3: Stripe Pre-Filter (SPRF)

Input: A set R of rectangles of size $a \times b$, threshold δ **Output:** Candidate regions R_c

```

1  $T \leftarrow \emptyset$ ;
2 Cut the whole space in horizontal direction into a set of stripes  $S$ ;
3 foreach  $s$  in  $S$  do
4    $R' \leftarrow$  Rectangles in  $R$  that intersect with  $s$ ;
5    $R'_c \leftarrow$  DARS Pre-Filter( $R'$ ,  $\delta$ );
6    $R_c \leftarrow R_c \cup R'_c$ ;
7 return  $R_c$ ;
```

Is it possible that we can reduce some computations so we do not need to rate *every* possible region?

The answer is positive. In Algorithm 1, if we find one maximal intersection, we need to check if it is already covered by other intersections or if it covers other intersections, and this is time-consuming. Therefore by first filtering with the δ -constraint we can reduce a lot of computation. We implement this filter idea as follows. In Algorithm 1, when sweeping (Line 1.2–1.11), we can first use the δ -constraint to filter out some groups of rectangles. This is because if a group of rectangles intersect, and their centers do not satisfy the δ -constraint, we can simply ignore this group of rectangles. We can simply add a checking process before Line 1.8. If centers of rectangles in G does not satisfy the δ -constraint, then we can terminate the current loop and start the next loop. In the end we will get some groups of rectangles and their corresponding maximal intersections, and the objects that each group contains will satisfy the δ -constraint. With these groups that satisfy the δ -constraint found, we then do not need to check δ -constraint when processing each maximal intersections (Line 2.11–2.15). Therefore we can remove the if check at Line 2.13.

With these modifications done to Algorithm 1 and Algorithm 2, we then have an optimized workflow of solving the DARS problem. We name this modified algorithm DPRF (DARS Pre-Filter).

3.3 Further Optimization with Stripes

As a matter of fact, for every possible group of intersecting rectangles, if we want to know if one such group satisfies δ -constraint, we need to scan this group, so we cannot just ignore it like BRS [7] does. Thus, our further optimization strategy needs to focus on optimizing the execution of the algorithms.

By dividing the space range in one dimension into a series of intervals, we can further reduce the computations needed to be done. Here we cut the space with vertical lines, so in the horizontal direction there will be a series of intervals. These intervals and the splitting vertical lines will form a series of rectangle spaces. We will denote as “stripes” these rectangle spaces. If there is a group of intersecting rectangles, the furthest distance between the leftmost vertical edge

and the rightmost vertical edge will always be less than $2 \times a$, otherwise they will not be intersecting each other, so we will set the width of each stripe to $2 \times a$. We illustrate this in Fig. 3. By doing so, a group of rectangles will only intersect with at most two stripes. Therefore, we can issue a new instance of DPRF for each stripe. Although the total number of loops will increase, things done in each loop will be reduced drastically compared to the increase in number of loops. We will see this effect in experiment section.

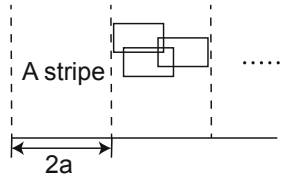


Fig. 3. Cutting space into stripes

We summarize the process in Algorithm 3. We issue a new DPRF instance for each stripe (Line 3.2–3.6, input to DPRF will be the rectangles that intersect with the stripe and the distance constraint δ , and these rectangles can be found by using the interval tree), then combine the results from each stripe and we will get the final results in the whole space. We call Algorithm 3 SPRF (Stripe Pre-Filter).

Table 2. Summary of datasets.

Property	BrightKite	Yelp	Synthetic
# of objects	772,966	157,776	60,000
Width	40,075 km	3,933 km	84 km
Height	20,015 km	2,012 km	152 km

Time Complexity Analysis. Suppose there are n rectangles, and denote the average size of active rectangles (A used in DPOF (Algorithm 2) and DPRF) as $|A|$, the average number of POIs in a region as $|r|$. It takes $O(|A||r|)$ time to check for maximal intersections, so the total complexity of DPOF is $O(n|A||r|)$. For DPRF, in the worst case it has the same complexity as DPOF (every possible group satisfies δ - constraint). As for SPRF, we construct an interval tree in $O(n \log n)$ time to get rectangles (Denoted as R' in Algorithm 3) that intersect with a stripe. It takes $O(\log n + |R'|)$ to find them with the interval tree ($|R'| \ll n$). There are at most $\frac{n}{2}$ stripe, and each DPRF instance has a complexity of $O(|R'| |r|^2)$. Thus, the total complexity of SPRF is $O(n \log n + \frac{n}{2}(\log n + |R'| + |R'| |r|^2)) = O(n \log n + n |R'| |r|^2)$.

4 Experiments

We first explain our setups for the experiments. Then we evaluate our methods on both quality and efficiency. Last, we run the scalability test.

4.1 Experiment Setup

Datasets. We use two real-life datasets BrightKite³ and Yelp⁴ to evaluate our methods. We will also use synthetic datasets for evaluation and scalability test. Summary of different datasets is in Table 2. POIs in BrightKite dataset are scattered over the whole globe. POIs in Yelp dataset are mainly in North America. POIs in synthetic dataset are generated in the scope of a normal city.

Query Rectangle Size. Different datasets come with different cardinality, and this will have impact on efficiency of the algorithms. Thus, we set different unit query rectangle size for different datasets. Given a dataset with cardinality $|O|$, we can get its minimum bounding rectangle of size $w \times h$, and the size of a unit query rectangle is $a \times b$, where $a = \frac{w}{|O|}$ and $b = \frac{h}{|O|}$. We can also apply a multiplier k to a and b so the query rectangle size becomes $ka \times kb$. Sometimes a or b might actually become a very small value (e.g., $2\text{ m} \times 1\text{ m}$) and this is irrational. Under such circumstances we will expand a or b until they become rational values, and we will use a and b after the expansion as the query unit size $q = a \times b$. We will indicate this expansion with different k in the figures. Different dataset has different k , hence leading to different sizes kq . In Fig. 4 we demonstrate how we choose a k value for Yelp dataset. For following experiments we choose $k = 8$ for Yelp dataset so that the region size is about $200\text{ m} \times 100\text{ m}$ which is proper. For other datasets, we omit reporting the figures due to space limit ($k = 4$ for BrightKite dataset and $k = 100$ for Synthetic dataset).

Performance Measures. We mainly focus on these measures: (a) Efficiency, which is the runtime of each algorithm. For SPRF, we set the width of each stripe to $2 \times a$ as discussed in Sect. 3.3. (b) The quality of returned regions that satisfy the δ -constraint. Quality is the value calculated according to f in Definition 1. Since DPOF, DPRF and SPRF achieve the same quality, we will use a united name DARS to compare with BRS. For BRS, we use f from Definition 1 for its pruning strategy. For evaluations involving no number of categories, we use $|C| = 3$. For evaluations involving no changing of δ -constraint, we use $\delta = \frac{a}{10}$ by default. Note that according to Definition 2 and 3, the smaller the quality is, the better the corresponding region is.

Since BRS [7] ignores a lot of possible groups of intersecting rectangles, it cannot guarantee that the result it finds satisfies δ -constraint (the result it finds might not be an answer to the DARS problem), so when compared with other methods, runtime is the main factor. But for the completeness we also compare BRS with other methods on quality.

³ <http://snap.stanford.edu/data/index.html>.

⁴ <https://www.yelp.com.sg/dataset/>.

Implementation. All algorithms are implemented in Java with JDK 11. We run all experiments on a PC with Intel i7-8700K 3.70 GHz CPU and 32 GB RAM.

4.2 Evaluation Results

For all the runtime evaluations we did not show the breakdown when reporting all efficiency results because the time taken for scoring a region was only 1%–2% compared to finding regions. Each experiment is repeated ten times, and the average result is reported.

Runtime (Different Datasets). First we evaluate the runtime of different algorithms on different datasets with proper k values. From Fig. 5 we can see that as $|O|$ grows, the runtime of DPOF increases fairly fast. It always scans and checks intersections without filtering with $\delta - constraint$. But it is also slightly faster than BRS and DPRF when the size of data is small (60,000 POIs in synthetic dataset), because DPOF does nothing but simply scanning and finding intersections, while BRS needs to first estimate an upper bound for each vertical interval, and DPRF needs to check $\delta - constraint$ whenever it runs into a possible group of rectangles. When the size of data is small, such as 60,000 POIs in synthetic dataset, these extra efforts in BRS and DPRF make them only a bit slower. But when the size of data grows, the advantages of BRS and DPRF stand out because they do not need to check intersections as often as DPOF does. The improved SPRF algorithm runs one magnitude faster than others on average.

Runtime (Different Query Rectangle Sizes). Figure 6 illustrates the runtime for different algorithms on different datasets with different query rectangle sizes. We find as the size of query rectangle grows, the runtime of each algorithm also grows because more intersections among rectangles are likely to appear. The runtime of DPOF grows still quite fast, while the runtime of BRS and DPRF grows slower. On the synthetic dataset with a relatively small size, DPOF still runs only a little faster because it does nothing during the scanning process except for finding the maximal intersections. SPRF is still the fastest here (1–2 magnitude faster).

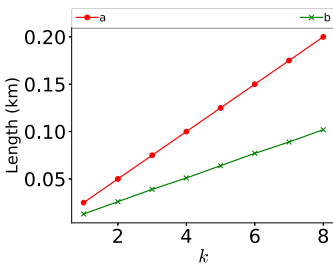


Fig. 4. a and b with different k .

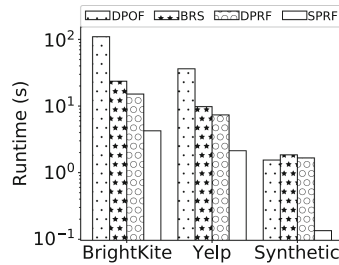


Fig. 5. Runtime on different dataset with proper k .

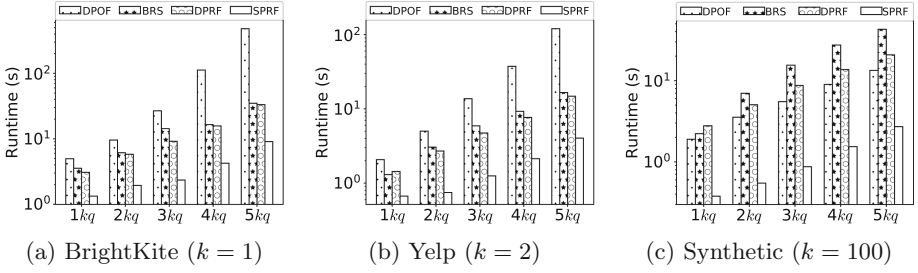


Fig. 6. Runtime with different query rectangle sizes.

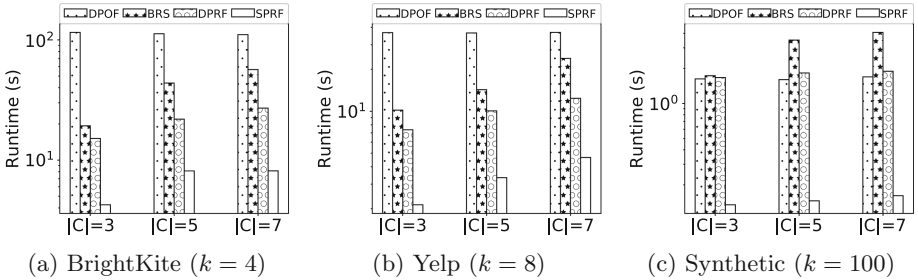


Fig. 7. Runtime with different number of categories.

Runtime (Different Number of Categories). We set different number of categories to evaluate its impact on the runtime of different algorithms in Fig. 7. As the number of categories grows, the runtime of DPOF stays almost the same and it is the slowest. As for the remaining algorithms, the runtime of each algorithm grows because more calculation needs to be done in $f(r)$. The growth on the runtime of DPRF and SPRF is the slowest because when there are more categories, there are fewer POIs of the same category needed to be checked against δ -constraint. Again SPRF outperforms the rest (1–2 magnitude faster).

Runtime (Different Constraints for DARS). In Fig. 8 we set different δ -constraints to evaluate the runtime of DPRF and SPRF. Since DPOF does not use δ -constraint during the process of scanning, the impact of different δ is tested only on DPRF and SPRF here. We set the constraint based on the width of a query rectangle by dividing the width with different values. We can see that as δ becomes smaller, the runtime of both algorithms decreases because when δ is small, fewer regions would satisfy the δ -constraint, so that fewer operations would be required to check new intersections and delete old intersections in Algorithm 1. SPRF is still faster than DPRF (3–10 times faster).

Quality (Different Query Rectangle Sizes). Because DPOF always finds the same answers as DPRF and SPRF do, we will use a united name DARS to compare with BRS. As DARS finds multiple top regions, we use the best one in order to compare with BRS because BRS always finds a single region as

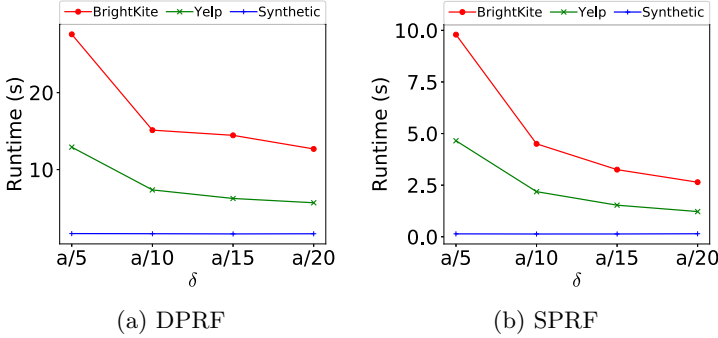


Fig. 8. DARS runtime with different δ – constraint.

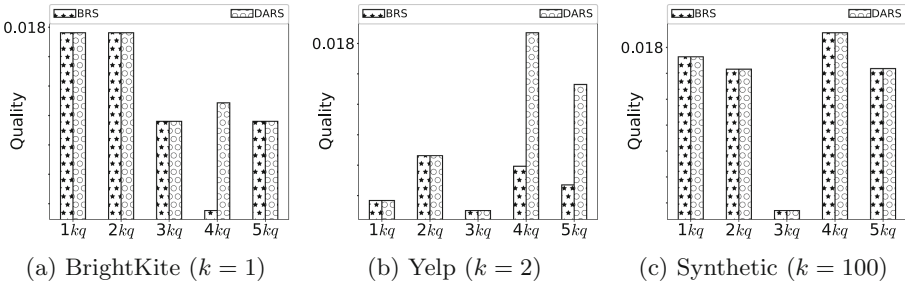


Fig. 9. Quality with different query rectangle sizes.

result. One thing about BRS is that it always finds the region that minimizes $f(r)$ without considering δ – constraint. So for $f(r)$, if BRS gets a value v_1 and DARS gets a value v_2 , $v_1 \leq v_2$ always holds. Now we set different query rectangle size for quality evaluations in Fig. 9. The difference between some values seems huge, but in fact the quality values differ at around 6–7 digits after the decimal point. For example, in Fig. 9a at $4kq$ the quality value from BRS is 0.018315693 while DARS returns a quality value of 0.018315712, and the quality value at $1kq$ is 0.018315724. This is the same for difference of quality values in Fig. 9b and Fig. 9c.

Quality (Different Number of Categories). In Fig. 10 we check the quality with different number of categories. As we can see difference brought by different $|C|$ is very obvious (due to how $f(O_r)$ works). When $|C| = 7$, quality returned from synthetic dataset does not reach a value as big as that from BrightKite and Yelp. Because in a relatively small dataset we might find regions with less than $|C|$ categories of POIs.

Quality (Different Constraints for DARS). We check the impact of different δ on the quality of a returned region with $|C| = 3$ by default. From Fig. 11 we can see that when the size of data is fairly small, the change in quality is obvious (the polyline of Synthetic dataset with stroke markers). When the size

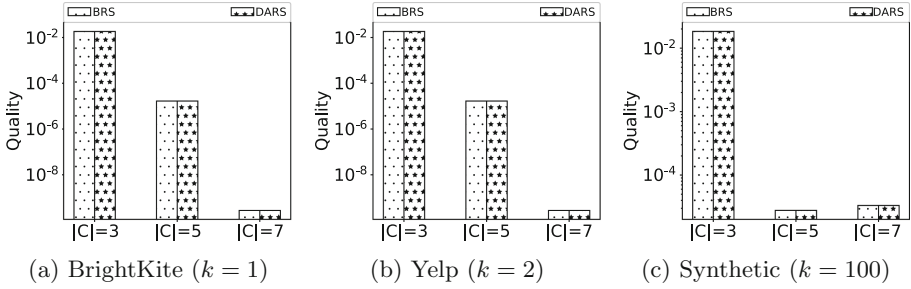


Fig. 10. Quality with different number of categories.

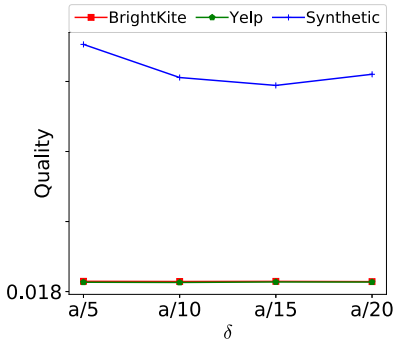


Fig. 11. DARS quality with different δ – constraint.

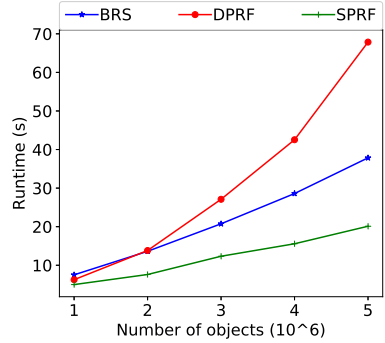


Fig. 12. Scalability test.

of data grows big, there is almost no change in quality because we are always likely to find all $|C|$ categories of POIs when there are more POIs no matter what the constraint value is.

Scalability Test. Now we generate synthetic datasets of different size to test the scalability of DPRF and SPRF because DPOF finds the same results and does not perform well on big dataset. We also test BRS here. We generate synthetic datasets in the same scope as in BrightKite dataset. The runtime is illustrated in Fig. 12. Neither DPRF nor BRS scales that well. SPRF scales a bit better, although when there are more than 3 million objects it still takes more than 10s to terminate, it can already handle real-life scenarios where there are not that many objects.

5 Related Work

The DARS problem is closely related to the *region search* problem. There are also other related studies such as the *maximizing range sum* problem [3]. Table 3 summarizes existing methods.

The Region Search Problem. Region search has been studied in the past few years. Feng et al. [7] studied the problem of *best region search* (BRS). Given a set O of all spatial objects, a rectangle r of a given size and a submodular and monotone function which extracts a property for a region (e.g., Number of categories in a region), BRS aims to identify a single “best region” that can maximize this function, and proposed algorithms and pruning techniques around the function. Another problem called *subjected-oriented top-k hot region query* (STR) was studied by Liu et al. [12]. They aim to find the top-k non-overlapping square regions that have the highest scores computed by the number of feature objects (e.g., objects of cultural features include museums, libraries, exhibition halls, etc.) and their weights.

Our DARS problem differs from these studies as follows. (1) We not only consider the number of categories, but also use the distance between POIs of different categories to consider an interrelationship, while in BRS only the number of categories was considered, and in STR this interrelationship was also omitted. (2) The interrelationship between POIs is unknown before the corresponding POIs are actually processed, and it remains a challenge to prune some POIs without affecting the correctness of the answers. Therefore, techniques in [7,12] cannot be directly applied to our DARS problem. (3) In STR the regions returned are non-overlapping, while in DARS the regions could overlap.

The Maximizing Range Sum Problem. The *Maximizing Range Sum* (MaxRS) problem [3] is also quite relevant to our work. First in the theoretical perspective for MaxRS Imai et al. [10] and Nandy et al. [14] aimed to find a rectangle to enclose the maximum number of points based on a classical *distribution-sweep* paradigm [9]. Given a set O of weighted points and a rectangle r of a given size, MaxRS aimed to find a rectangle range of the same size that maximizes the sum of weights of all the points covered by the rectangle (e.g., sum of influence of all points). A naive solution is to issue an infinite number of *range aggregate* (RA) operations [1, 11, 15–17] but this is prohibitively expensive. Choi et al. [3] solved the MaxRS problem with a scalable method in spatial databases.

Our DARS problem differs from MaxRS in these aspects: (1) we take into account the interrelationship between points in a region; (2) our objective function completely differs from a simple aggregate function in MaxRS (e.g., SUM, COUNT). As a result, DARS does not consider a weight for each POI while MaxRS does.

There are also other studies less relevant to this work. To name a few, some studies tried to capture the features of regions and find similar regions to a given query region [8,13]. Region-wise deployment of a set of facilities or advertisements are studied in [21,22], aiming to maximize the number of users influenced. Choi et al. [2] proposed *nearest neighbourhood search* to find circular nearest regions that contains some number of POIs through pure geometric computations.

Table 3. Summary of methods

Methods	Diversity	Interrelationship	Objective function
<i>BRS</i>	#Category	None	Submodular and monotone
<i>MaxRS</i>	#Category	None	Aggregate
<i>DARS</i>	#Category	Distance between POIs	Not submodular, not aggregate

6 Conclusions

In this paper we introduced a novel problem called the *diversity and distribution-aware region search* (DARS). The goal of DARS is to find regions that are diversified (contain many categories of POIs) and the POIs within these regions are well-distributed so that users can enjoy several kinds of services they want without spending much time on commuting. We proposed several methods including the baseline method DPOF, an improved method DPRF and a further optimized method SPRF. SPRF can handle many kinds of scenarios quite efficiently in extensive experiments over real-world datasets. In the future we would like to discuss how to solve the DARS problem in a road network environment, or assigning different weights to different categories.

Acknowledgements.. Special thanks to Kunkui Yang at BaiZhi Data Technology Co., Ltd., Nanjing, China for providing essential inspirations. This research is supported in part by ARC DP200102611, DP180102050, NSFC 91646204.

References

1. Cho, H., Chung, C.: Indexing range sum queries in spatio-temporal databases. *Inf. Softw. Technol.* **49**(4), 324–331 (2007)
2. Choi, D., Chung, C.: Nearest neighborhood search in spatial databases. In: *ICDE*, pp. 699–710. IEEE Computer Society (2015)
3. Choi, D., Chung, C., Tao, Y.: A scalable algorithm for maximizing range sum in spatial databases. *PVLDB* **5**(11), 1088–1099 (2012)
4. Choudhury, F.M., Culpepper, J.S., Bao, Z., Sellis, T.: Finding the optimal location and keywords in obstructed and unobstructed space. *VLDB J.* **27**(4), 445–470 (2018). <https://doi.org/10.1007/s00778-018-0504-y>
5. Drezner, Z., Hamacher, H.W.: *Facility Location - Applications and Theory*. Springer, Heidelberg (2002)
6. Du, Y., Zhang, D., Xia, T.: The optimal-location query. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) *SSTD 2005*. LNCS, vol. 3633, pp. 163–180. Springer, Heidelberg (2005). https://doi.org/10.1007/11535331_10
7. Feng, K., Cong, G., Bhowmick, S.S., Peng, W., Miao, C.: Towards best region search for data exploration. In: *SIGMOD Conference*, pp. 1055–1070. ACM (2016)
8. Feng, K., Cong, G., Jensen, C.S., Guo, T.: Finding attribute-aware similar region for data analysis. *PVLDB* **12**(11), 1414–1426 (2019)
9. Goodrich, M.T., Tsay, J., Vengroff, D.E., Vitter, J.S.: External-memory computational geometry (preliminary version). In: *FOCS*, pp. 714–723. IEEE Computer Society (1993)

10. Imai, H., Asano, T.: Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms* **4**(4), 310–323 (1983)
11. Lazaridis, I., Mehrotra, S.: Progressive approximate aggregate queries with a multi-resolution tree structure. In: *SIGMOD Conference*, pp. 401–412. ACM (2001)
12. Liu, J., Yu, G., Sun, H.: Subject-oriented top-k hot region queries in spatial dataset. In: *CIKM*, pp. 2409–2412. ACM (2011)
13. Liu, Y., Zhao, K., Cong, G.: Efficient similar region search with deep metric learning. In: *KDD*, pp. 1850–1859. ACM (2018)
14. Nandy, S., Bhattacharya, B.: A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Comput. Math. Appl.* **29**(8), 45–61 (1995)
15. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) *SSTD 2001*. LNCS, vol. 2121, pp. 443–459. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-47724-1_23
16. Sheng, C., Tao, Y.: New results on two-dimensional orthogonal range aggregation in external memory. In: *PODS*, pp. 129–139. ACM (2011)
17. Tao, Y., Papadias, D.: Range aggregate processing in spatial databases. *IEEE Trans. Knowl. Data Eng.* **16**(12), 1555–1570 (2004)
18. Wong, R.C., Özsu, M.T., Yu, P.S., Fu, A.W., Liu, L.: Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB* **2**(1), 1126–1137 (2009)
19. Xiao, X., Yao, B., Li, F.: Optimal location queries in road network databases. In: *ICDE*, pp. 804–815. IEEE Computer Society (2011)
20. Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive computation of the min-dist optimal-location query. In: *VLDB*, pp. 643–654. ACM (2006)
21. Zhang, P., Bao, Z., Li, Y., Li, G., Zhang, Y., Peng, Z.: Trajectory-driven influential billboard placement. In: *KDD*, pp. 2748–2757. ACM (2018)
22. Zhang, Y., Li, Y., Bao, Z., Mo, S., Zhang, P.: Optimizing impression counts for outdoor advertising. In: *KDD*, pp. 1205–1215. ACM (2019)