Efficient Progressive and Diversified Top-k Best Region Search

Dimitrios Skoutas
Athena R.C.
dskoutas@imis.athena-innovation.gr

Dimitris Sacharidis
TU Wien
dimitris@ec.tuwien.ac.at

Kostas Patroumpas
Athena R.C.
kpatro@imis.athena-innovation.gr

ABSTRACT

Given a set of geospatial objects, the *Best Region Search* problem finds the optimal placement of a fixed-size rectangle so that the value of a user-defined utility function over the enclosed objects is maximized. The existing algorithm for this problem computes only the top result. However, this is often quite restrictive in practice and falls short in providing sufficient insight about the dataset. In this paper, we introduce the k-BRS problem, and we present a method for efficiently and progressively computing the next best result for any number of results k requested by the user. We show that our approach can accommodate additional constraints. In particular, we consider the requirement of computing the next best rectangle that has no or little overlap with the already retrieved ones, which reduces the repetition and redundancy in the results presented to the user. Our experimental evaluation demonstrates that our algorithms are efficient and scalable to large real-world datasets.

CCS CONCEPTS

• **Information systems → Geographic information systems**;

KEYWORDS

best region search, areas of interest, spatial analytics

1 INTRODUCTION

Geospatial data continue to grow daily, offering a wealth of information about locations and human activities. This can be exploited in various domains, from geomarketing and tourism to real estate and urban planning, allowing to identify *areas of interest* and thus drive, for instance, decisions about selecting the best location to open a new store or to place an advertisement.

The *Maximizing Range Sum* (MaxRS) problem [4] assumes a set of spatial objects as weighted points in a 2D space and aims at identifying the optimal location of a fixed-size rectangle that maximizes the total weight of the enclosed points. Different variations of the problem have been studied in the past by the computational geometry community [10, 13] and, due to the increasing data volumes, have also gained focus more recently among the database community [1, 2, 4, 6–8, 15, 17, 18]. In particular, the *Best Region Search* (BRS) problem [6] generalizes MaxRS by allowing the scoring function to be any submodular monotone function rather than being restricted to the *count* or *sum of weights* of the enclosed points.

The BRS problem has numerous applications. Consider, for example, a company wishing to open a store at a new location. A rectangle of a given size centered at the store's candidate location can be used to approximate the area from which the store is expected to draw its potential customers. Then, the score of a location can be measured by an objective function that could be simply the number of potential customers in the area or some more complex function encapsulating various revenue estimation criteria. A similar example is a tourist looking for the optimal location of a hotel, aiming to maximize the number or popularity of sites that can be visited in its vicinity.

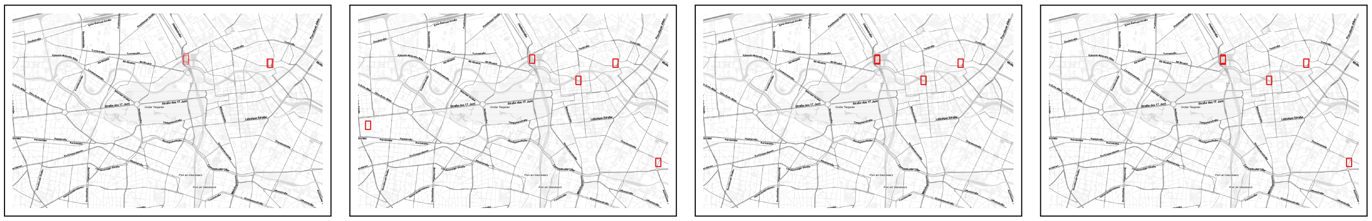
However, an inherent limitation of existing formulations and solutions is that they explicitly target the *best*, i.e., the top-1, answer to the query. In many practical scenarios and applications this is not sufficient. For instance, in the aforementioned examples, it may not be possible to open a store at the identified best location (e.g., if there are no available facilities to rent or purchase); similarly, all hotels in the area may be occupied or too expensive. Then, the user needs to examine alternative solutions in decreasing order of quality, until one is found that meets all desired criteria.

In this paper, we introduce and study the *k-Best Region Search* (k-BRS) problem, which searches for a ranked list of the top-k best rectangles according to the objective score function. As explained later, existing algorithms for the BRS problem do not apply for the k-BRS problem. Furthermore, we observe that returning a ranked list of results based on their objective score still faces an important shortcoming, namely that the results are usually highly overlapping. For instance, if R is the best result, then slightly shifting R along the x- and/or the y-axis leads to another candidate result R' that encloses approximately the same set of objects, and is thus very likely to have a similarly high score. Nevertheless, if R has already been found to not meet the user's other criteria or preferences, the same is also very likely to hold for R'. Returning results that highly overlap with already seen ones, is most likely redundant.

As a concrete example, consider finding the top rectangles over a set of Points of Interest in Berlin. Assume that we are interested in regions of width and height 0.002° (~ 200 meters), and that the scoring function is *count*. Figure 1a shows the top-100 rectangles enclosing the largest number of points. In fact, even though 100 results are retrieved, they essentially draw attention to merely two locations: there are 26 highly overlapping rectangles in the left group of results and 74 in the other.

To address this redundancy issue of the unrestricted k-BRS, we need to select the top-k results in a way that considers not the individual objective score of each candidate result in isolation, but rather its *marginal gain*. This refers to the added value of a new result in the context of those already selected. We note that similar

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274965



(a) Top-100 unrestricted. **(b) Top-5 in NO mode.** **(c) Top-5 in PO mode ($\lambda = 0.2$).** **(d) Top-5 in PO mode ($\lambda = 0.3$).**

Figure 1: Example of top-k regions with different constraints for overlap.

diversification approaches have been taken in information retrieval [3, 5, 9, 16] and summarization of spatio-textual objects [12, 14].

To that end, we introduce and study the *k-Best Region Search with Marginal Gain* (k-BRS-MG) problem. Specifically, we consider two variants of this problem. In the *No Overlap (NO)* case, the results in the top-k list are not allowed to overlap with each other. An example of the output produced under this setting is shown in Figure 1b, where the top-5 results are depicted. Notice that these include the two locations identified before, but also reveal three additional locations that may be of interest to the user. The second variant, which we call *Partial Overlap (PO)*, offers a user-tuneable tradeoff between objective score and allowed overlap in the results. Specifically, PO uses an exponential decay function to discount the objective score of candidate results based on their degree of overlap with already retrieved results. The user can set the value of a single parameter $\lambda > 0$ to control the rate of decay. Smaller values of λ produces results similar to the case of unrestricted k-BRS, while larger values favor results similar to the NO case.

Results produced under the PO setting are shown in Figures 1c and 1d. When λ is set to 0.2, i.e., a relatively small penalty for overlap, the results indicate three distinct locations with the one on the left having three overlapping rectangles. When λ is increased to 0.3, one of the overlapping results is discarded in favor of another, leading to identifying four distinct locations. This shows that by tuning the decay parameter λ we can get more or less diversified results in the spectrum of completely allowing overlap to completely prohibiting overlap.

Our contributions can be briefly summarized below.

- We introduce and formally define the problem of k-Best Region Search (k-BRS), and we present an efficient progressive algorithm for it.
- We introduce the problem of k-Best Region Search with Marginal Gain (k-BRS-MG) that produces diversified results. We discuss two variants of the problem that differ in the way they penalize overlaps among results.
- We extend our top-k progressive algorithm for solving the two variants of the k-BRS-MG problem.
- We present an experimental evaluation of the proposed algorithms using three real-world datasets.

The rest of the paper is structured as follows. Section 2 reviews related work focusing on the MaxRS and BRS problems. Section 3 defines the two problems (k-BRS and k-BRS-MG), and explains the challenges involved. Then, Sections 4 and 5 present our solutions to these problems. Section 6 presents our experimental study, while Section 7 concludes the paper.

2 RELATED WORK

Region Search Problems. Various definitions of what constitutes an *area of interest* in a dataset of spatial objects are possible (e.g., [11]), leading to several classes of problems. One such family of problems considers areas of interest represented by *rectangles of fixed width and height*. Each rectangle is assigned an objective score based on the set of points it encloses.

The simplest variant is to compare different rectangles based on the number of points they contain. This is known as the *Maximum Object Enclosing Rectangle* problem, for which a line-sweep-based algorithm utilizing an interval tree data structure has been proposed [10, 13]. More recently, the *Maximizing Range Sum* (MaxRS) problem has been studied [4]. In this case, a set of weighted points is given, and the goal is to find a rectangular area of fixed size, such that the sum of the weights of all enclosed points is maximized. The proposed solution focuses on an external-memory algorithm that performs an external version of the plane-sweep algorithm, recursively dividing the entire dataset into smaller sets until they fit in memory. Furthermore, an $(1 - \epsilon)$-approximate solution for the MaxRS problem has been presented in [15].

The *Best Region Search* (BRS) problem has been introduced in [6] as a generalization of the MaxRS problem. The difference is that the objective score function used to compute the utility of a rectangle can be any submodular monotone function. Thus, problem variants where this function is *count* or *sum* can be regarded as specific instances of this problem formulation. In this paper, we follow this more generic problem formulation in our definition of the top-k Best Region Search problem. As the algorithm in [6] is crucial in explaining our methodology, we later present it in detail.

Various other extensions to these problems have been studied in the literature. Indicatively, variants of the MaxRS problem have been considered in road networks [2, 18], where the region to be identified is defined as a subgraph that does not exceed a given size constraint. Also, similar problems have been investigated under *bichromatic* settings, where two distinct datasets are given as input, i.e., a set of *customers* and a set of *facilities*, and the goal is to find the optimal location for a new facility to maximize the served customers considering also the existing facilities (e.g., [17]). In a streaming context, the work in [1] presents a generic framework for *continuous MaxRS* monitoring based on a branch-and-bound approximation algorithm with worst-error guarantees. In a similar direction, *Top-k Bursty Regions* [7, 8] has been proposed as a continuous query for the top-k rectangles against streaming spatial objects.

Best Region Search. The BRS algorithm was proposed in [6]. Each point is represented by a fixed-size rectangle centered at it. Finding

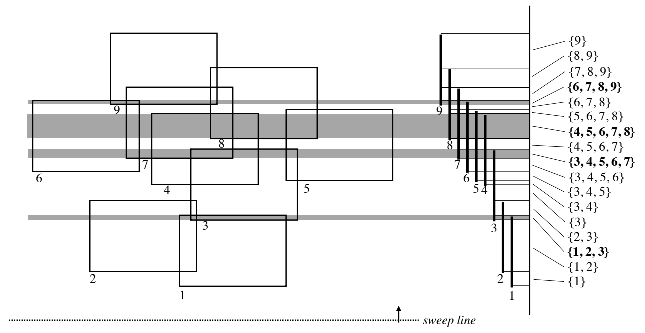


Figure 2: Bottom-up sweep and maximal slabs.

the best location to place the result rectangle entails identifying the regions that are maximal intersections of these rectangles. The intuition is that if we take a result rectangle centered within such a maximal region, it will cover a maximal set of points (the rectangles' centers), and by the property of the scoring function, it will have a higher objective score than any other result rectangle in its vicinity.

BRS partitions the input space in vertical *slices* that run parallel to the y-axis. Each slice is processed by executing a *bottom-up scan* over it using a *horizontal sweep line*. This line moves upwards until it encounters the *bottom* or the *top* edge of a point's rectangle, meaning that it *enters* or *exits*, respectively, the range of a point. Exploiting the monotonicity of f, the sweep line continues to move upwards as long as it encounters bottom edges of rectangles (i.e., new points enter its range), and only stops when the top edge of a rectangle is found (i.e., one of these points now exits its range). The portion of space between the last encountered bottom edge and the first encountered top edge constitute a so-called *maximal slab*.

Figure 2 presents an example with 9 rectangles, where the maximal slabs discovered by the sweep line are shaded. This bottom-up scan is equivalent to finding the maximal intersections of the 1D intervals that correspond to the y-extents of the rectangles (depicted with bold). By the properties of function f, the best rectangle must cover a maximal set of points, and thus its center must be positioned within one of the maximal slabs. Moreover, an upper bound for the score of any rectangle centered within a maximal slab can be obtained by computing the value of f over the respective set of points encountered along the way of producing this maximal slab.

The maximal slabs are visited in decreasing order of their upper bound. They are processed similarly to slices, but being scanned from *left to right* using a *vertical sweep line*. This scan produces a set of *maximal regions* (not to be confused with the regions, i.e., the rectangles, to be returned as results). Like maximal slabs, these have the property that the best rectangle is guaranteed to be centered in one of those maximal regions. Moreover, the score of a rectangle centered within a maximal region is equal to the value of f over the points corresponding to that maximal region. Once a maximal region is found, the score of the respective rectangle centered in it is compared to the current best rectangle. If it is higher, it becomes the new solution, otherwise it is discarded and the search continues.

3 PROBLEM DEFINITION AND CHALLENGES

We first introduce the main concepts and notation used throughout the paper. Then, we formally define the problem addressed, and we explain its challenges.

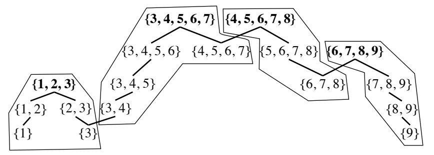


Figure 3: Slab relationships encoded as a DAG.

3.1 Problem Definition

Assume a set $\mathcal{D}$ of points in a two-dimensional space. These may represent Points of Interest on a map, or any other kind of spatial object. Each point may be further associated with an arbitrary number and type of attributes, such as a score denoting its importance or popularity (e.g., a rating or the number of visits, likes, views, etc.), keywords or tags representing its type or amenities and services offered, and so on.

Definition 1 (Region). *A region R is a rectangle of fixed width a and height b. We use $R.center$ to refer to the center of the region.*

Note that a region R can be positioned freely anywhere in space, i.e., its center $R.center$ is not restricted to be one of the points in $\mathcal{D}$. Without loss of generality, in the rest of the paper we assume regions of equal width and height, thus simplifying notation by referring to a single input parameter $a = b = \varepsilon$ instead of two distinct parameters a and b. Also, we often use the terms region and rectangle interchangeably. Furthermore, we use $R.P$ to denote the set of points that are contained within region R, i.e.,

$$R.P = \{p \in \mathcal{D} : p.within(R)\}$$

Definition 2 (Objective Score). *Assume a* scoring function *$f : P \longrightarrow \mathbb{R}$ that assigns a score $f(P)$ to a given set of points $P \subseteq \mathcal{D}$. The* objective score *of a region R, denoted as $R.score$, is the score of the set of points it contains, i.e.,*

$$R.score = f(R.P)$$

The function f can be application-specific, measuring the *utility* of a region based on its contents. A typical example is an aggregate function that measures the total number or the total score of the points within the region. Another example is a function that measures the number of distinct keywords associated to the points within the region, e.g., as a way to quantify the variety of amenities and services offered in it.

In this paper, we consider scoring functions that are *monotone*. This means that by adding more points to a given set, the objective score of the set increases (or remains the same), i.e., for any set of points $P \subseteq \mathcal{D}$ and any additional point $p \in \mathcal{D}$ it holds that:

$$f(P) \leq f(P \cup \{p\})$$

This assumption is quite reasonable; for example, all aforementioned scoring functions are monotone.

Notice that, since a region can be positioned freely anywhere in space, there can be an infinite number of regions of the same size that contain the same set of points. To avoid this ambiguity, we consider the *canonical* region of a set of points P to be the one that is centered at the center of the minimum bounding rectangle (MBR) of P. This allows for a one-to-one correspondence between a region R and the set of points $R.P$ it contains, ensuring that the search space of candidate regions is finite (bounded by the size of the power set of $\mathcal{D}$) and does not contain multiple areas with exactly the same contents. Thus, for the rest of the paper, we only consider canonical regions.

We can now define the *k-Best Region Search problem.*

Problem 1 (k-BRS). *Given a set of points $\mathcal{D}$, a monotone scoring function f, and a region width and height ε, the k-BRS problem is to find the top-k regions having the maximum objective score, i.e., to compute an ordered list $\mathcal{L}$ of k regions such that:*

$$\forall i \in [1,k] : \mathcal{L}_i = \arg \max_{R \in \mathcal{R} \setminus \mathcal{L}_{1:i}} f(R.P)$$

where $\mathcal{L}_i$ denotes the i-th element of $\mathcal{L}$, $\mathcal{L}_{1:i}$ denotes the subset of $\mathcal{L}$ containing the first $i-1$ elements, and $\mathcal{R}$ is the universe of all canonical regions.

Even by considering only canonical regions, the results returned by k-BRS can be highly overlapping. To address this, we introduce a variant that explicitly takes into account overlaps among regions. That is, we need to select the top-k results in a way that considers not only the individual objective score of each candidate region, in isolation, but rather the added value of each new region in the context of those already selected. More specifically, instead of ranking the regions based on their individual objective score, we use the notion of *marginal gain*, an adjusted score that is computed taking into consideration the already seen results. This requires an incremental process for building the top-k result set, by starting from the top-1 result and at each iteration selecting the i-th result to be the one that has the maximum marginal gain with respect to the previous $i-1$ results.

Definition 3 (Marginal Gain). *Given a list $\mathcal{L}$ of regions, the* marginal gain (MG) *of a region $R \notin \mathcal{L}$ is defined as:*

$$mg(R; \mathcal{L}) = \gamma(R; \mathcal{L}) \times f(R.P)$$

where $\gamma(R; \mathcal{L})$ quantifies the novelty *of R with respect to $\mathcal{L}$.*

We now present the k-BRS problem variant that retrieves the top-k best regions based on their marginal gain rather than their objective score.

Problem 2 (k-BRS-MG). *Given a set of points $\mathcal{D}$, a monotone scoring function f, and a region width and height ε, the* k-BRS-MG *problem is to find the top-k regions having the maximum marginal gain score, i.e., to compute an ordered list $\mathcal{L}$ of k regions such that:*

$$\forall i \in [1,k] : \mathcal{L}_i = \arg \max_{R \in \mathcal{R} \setminus \mathcal{L}_{1:i}} mg(R; \mathcal{L}_{1:i})$$

where $\mathcal{L}_i$, $\mathcal{L}_{1:i}$, and $\mathcal{R}$ as above.

We propose two flavors of the k-BRS-MG problem that employ different novelty functions. The *No Overlap (NO)* variant completely prohibits overlapping regions among the top-k results. In this case, function γ is defined as follows:

$$\gamma_{\text{NO}}(R; \mathcal{L}) = \begin{cases} 1, & \text{if } \nexists R' \in \mathcal{L} : R.\text{overlaps}(R') \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The *Partial Overlap (PO)* variant discounts the objective score of a candidate region R based on its degree of overlap with the existing results in $\mathcal{L}$. We model this using exponential decay. Specifically, in this case, we define function γ as follows:

$$\gamma_{\text{PO}}(R; \mathcal{L}) = e^{-\lambda \cdot \tau(R; \mathcal{L})} \tag{2}$$

where $\lambda > 0$ is a parameter controlling the rate of decay and $\tau(R; \mathcal{L})$ measures the maximum degree of overlap between R and any of the results in $\mathcal{L}$, i.e.:

$$\tau(R; \mathcal{L}) = \max_{R' in \mathcal{L}} \frac{area\_size(R.intersection(R'))}{area\_size(R)}$$

3.2 Challenges

The k-BRS problem and its marginal gain variants cannot be solved using the algorithm introduced in [6]. There, the problem is to identify the *best* region, i.e., top-1 instead of top-k. Adapting the process to retrieve top-k results instead of only the top-1 is not straightforward. This is because the basic idea of the BRS algorithm relies on the fact that the objective score function f is a monotone function, which implicitly leads to the algorithm being designed and optimized under the assumption that only the top-1 result needs to be computed. Essentially, the efficiency of the algorithm is based on the fact that, during the search, it can *skip over* regions that cannot be in the top-1 rank. However, by doing so, once the top-1 result is identified, it is impossible to continue the search onwards to retrieve the top-2 result, and so on.

To better understand this, recall that each maximal slab or region is preceded and succeeded by a series of other slabs or regions, respectively. When only the top-1 result is needed, the sweep line can safely skip those and only examine the maximal one, since it is certain that its score will be higher. For example, in Figure 2, BRS only examines candidate results that are centered within the shaded areas of the space. Essentially, we can think of maximal slabs and regions as the positions of local maxima in the search space. It is guaranteed that the global maximum will be one of those. However, when top-k results are needed, with $k > 1$, solutions around a local maximum cannot be ignored, since they may be (and very often are) better than those provided by other local maxima. The implication is that to correctly identify the top-k results the algorithm needs to be able to expand the search to include the space around the local maxima so that no valid solutions are missed.

Finally, another drawback of the BRS algorithm has to do with the way it partitions the space. Specifically, it partitions the space across only one dimension, in particular across the x-axis, creating vertical *slices* that run in parallel to the y-axis. The width of a slice can be controlled by a parameter and can be set to be as small as the width of a region; however, the height of a slice is equal to the height of the whole space occupied by the points in the dataset. This may be sufficient for small datasets where the height of the whole space is not much larger than the height of a region. This is often not the case in practice. For example, the user may wish to identify top-k regions having size in the order of a few building blocks within a large metropolitan area or even a whole country. Moreover, often in real-world spatial datasets the points do not follow a uniform spatial distribution but instead are rather skewed, with certain locations having a high density of points while others being very sparse. Partitioning the dataset only across one dimension typically fails to reflect this skewness in the distribution of the points.

Algorithm 1: k-BRS

```
Input: Set of points $\mathcal{D}$; Objective score function $f$; Region width and height
    $\varepsilon$; Number of results $k$
Output: Ranked list $\mathcal{L}$ of top-$k$ regions
▷ Initialization
1  $Q \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$
2  $\mathcal{G} \leftarrow$ CreateGrid($\mathcal{D}$, $\varepsilon$)
3  for $C \in \mathcal{G}$ do
4      $UB(C) \leftarrow$ ComputeCellUpperBound($C$, $neighbors(C)$, $f$)
5      $Q$.add($C$)
6  while $|\mathcal{L}| < k$ & $|Q| > 0$ do
7      $E \leftarrow Q.next()$
       ▷ Processing a Cell
8      if $E$.type = Cell then
9          $\mathcal{R}_E \leftarrow$ GetRectangles($E$)
10         $\mathcal{R}_E \leftarrow$ sort($\mathcal{R}_E$, $Y$)
11         $\mathcal{S} \leftarrow$ Sweep($\mathcal{R}_E$)
12         $Q$.addAll($\mathcal{S}$)
       ▷ Processing a Slab
13     if $E$.type = Slab then
14         $\mathcal{R}_E \leftarrow$ GetRectangles($E$)
15         $\mathcal{R}_E \leftarrow$ sort($\mathcal{R}_E$, $X$)
16         $\mathcal{T} \leftarrow$ Sweep($\mathcal{R}_E$)
17         $Q$.addAll($\mathcal{T}$)
18         $\mathcal{S}' \leftarrow$ GetChildren(S)
19         $Q$.addAll($\mathcal{S}'$)
       ▷ Processing a Region
20     if $E$.type = Region then
21         $R^{\varepsilon} \leftarrow$ CatchmentArea($E$)
22         $\mathcal{L}$.add($R^{\varepsilon}$)
23         $\mathcal{T}' \leftarrow$ GetChildren(T)
24         $Q$.addAll($\mathcal{T}'$)
25 return $\mathcal{L}$
```

Algorithm 2: Sweep

```
Input: Set of rectangle edges $\mathcal{R}_E$
Output: Set of slabs $\mathcal{S}$
1  $\mathcal{S} \leftarrow \varnothing$; $P \leftarrow \varnothing$
2  previous ← close
3  for each edge $R.e$ in $\mathcal{R}_E$ do
4      if $R.e$ is open then
5          if previous is close then
6              $S \leftarrow$ new slab; $S$.before = 0; $S$.after = 0; $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$
7              previous ← open
8          else
9              $S$.before++
10         $P \leftarrow P \cup \{R\}$
11     if $R.e$ is close then
12         if previous is open then
13             $S.P \leftarrow P$
14             previous ← close
15         $S$.after++
16         $P \leftarrow P \setminus \{R\}$
17 return $\mathcal{S}$
```

4 ALGORITHM FOR k-BRS

Next, we present our algorithm for the k-BRS problem. Its key idea is to sweep the space and examine subareas (e.g., slabs and regions) besides the maximal ones, as this is necessary to identify the next best region. However, in contrast to an *exhaustive* search that considers all subareas, e.g., all slabs in Figure 2, the k-BRS algorithm examines slabs in a principled way, considering one only when necessary.

The k-BRS algorithm, shown in Algorithm 1, consists of an initialization phase, and a main loop where elements from a priority queue are dequeued and processed. There are three types of elements (cells, slabs, and regions) and their processing differs. In what follows and similar to BRS, we represent each point in the input dataset as an $\varepsilon \times \varepsilon$ rectangle centered around the point.

Initialization. Our k-BRS algorithm employs a different initialization phase than BRS. Instead of partitioning the space in one dimension, i.e., in a series of vertical slices running parallel to the y-axis, it constructs a uniform grid with cells of size $\varepsilon \times \varepsilon$ (Line 2). By doing so, it partitions the space across both dimensions and with a resolution that matches that of the regions to be computed. In this way, it adapts to the spatial distribution of the input dataset better, allowing the search to focus around the most promising cells while pruning early other portions of the space that cannot produce regions with sufficiently high score.

For each cell C in the grid, we can derive an upper bound $UB(C)$ for the objective score of any region R centered within C (Line 4). Let $C.P$ denote the points inside cell C. Then, $UB(C)$ can be obtained by computing the value of f over the union of points in C and its eight adjacent cells, denoted by $neighbors(C)$, as follows:

$$UB(C) = f\left(\cup_{C' \in \mathcal{N}_C} C'.P\right)$$

where $\mathcal{N}_C = \{C\} \cup neighbors(C)$. The above upper bound holds because any region centered within C may contain at most those points located inside C and its neighboring cells. Note that it is possible to derive a tighter upper bound by only considering those points in the neighboring cells that are located within a buffer of width $\varepsilon/2$ from the borders of C. This implies a trade-off between the overhead of this extra filtering step and any resulting time savings from having a tighter upper bound. In either case, this does not affect the correctness of the algorithm.

Each cell is then added to a priority queue Q in descending order of its upper bound (Line 5). This priority queue is used throughout the algorithm to navigate the search space of candidate solutions, allowing to retrieve the top-k results *progressively*. To achieve this, Q holds two other types of entries, besides cells. Reusing the terminology of the BRS algorithm, these other types of entries are called *slabs* and *regions*, which are generated and inserted in the queue as will be explained later on. Every time the next element E is extracted from Q, the algorithm checks its type, i.e., whether it is a cell, a slab, or a region, and processes it accordingly.

Processing a Cell. Processing a cell C essentially works in a similar way as processing a slice in BRS. Let $\mathcal{R}_C$ denote the set of $\varepsilon \times \varepsilon$ rectangles corresponding to the points in C and its neighbors (Line 9). These are sorted bottom-up according to their y-coordinate (Line 10). Then, a horizontal sweep line is used to scan the cell bottom-up (Line 11). During this scan, the set of maximal slabs $\mathcal{S}$ is identified and emitted. For each maximal slab, the algorithm computes its corresponding upper bound $UB(S) = f(S.P)$, and inserts the maximal slab in the Q with key its upper bound (Line 12).

In addition, and contrary to BRS, additional information is maintained per slab so that the search can continue beyond the maximal slabs, when necessary. Assume that the rectangles of Figure 2 correspond to the points of a cell and its neighbors. As discussed, the sweep line identifies the intersections of rectangles, or slabs, along the y-axis also depicted. Observe that all slabs can be structured as the directed acyclic graph (DAG) depicted in Figure 3. An edge between two slabs exists if they touch each other, or equivalently if their intersection sets differ by one element (a rectangle). Moreover, a slab is a parent of another if the former's intersection set contains the latter's. Clearly, maximal slabs, shown bold, have no parents.

The top-k BRS algorithm stores within each maximal slab a partition of the DAG, which we call *slab tree*, that has the following properties: (i) it is a binary tree rooted at a maximal slab, (ii) each node other than the root has at most one child, (iii) the leftmost leaf (before the root in the order of the sweep line) corresponds to a slab that either has no child or its child has another parent in the DAG, and (iv) the rightmost leaf (after the root in the order of the sweep line) corresponds to a slab that has no child in the DAG. The partition of the DAG into slab trees is shown in Figure 3.

An efficient way to encode a slab tree for S is to store the points of the maximal slab in a list $S.P$, sorted in the order they were encountered by the sweep line, along with two counters, S.before, S.after, that indicate the number of children before and after S in the order of the sweep line. For example, in Figure 3, the maximal slab $\{3, 4, 5, 6, 7\}$ is encoded by its list of points $S.P = \{3, 4, 5, 6, 7\}$, and counters $S.\text{before} = 3$, $S.\text{after} = 1$.

Algorithm 2 presents in detail the Sweep procedure that scans a set of rectangles and creates the maximal slabs and their associated slab trees. Its operation is easily understood in the context of the DAG encoding of the slab relationships, e.g., Figure 3. Conceptually, Algorithm 2 scans the DAG from left to right. When a rectangle opens (lines 4–10), i.e., its first edge is encountered, we are moving up on the DAG, while when a rectangle closes (lines 11–16), i.e., its second edge is encountered, we are moving down on the DAG. When we move up for the first time since moving down (lines 5–7), a new slab tree is generated (line 6) where the maximal slab will be later determined. On the other hand, when we move down for the first time since moving up (lines 11–14), the maximal slab is identified.

Processing a Slab. Whenever a slab S is extracted from Q, its processing involves now two actions. The first action is similar to scanning a cell, but is now performed along the x-axis. That is, the set of rectangles $\mathcal{R}_S$ corresponding to the points in the slab is constructed and sorted according to their x-coordinate (Lines 14 and 15). Then, a vertical sweep line is used to scan the slab from left to right, identifying along the way the set of all maximal regions $\mathcal{T}$. Each maximal region $T \in \mathcal{T}$ is associated to a set of points $T.P$ and a corresponding upper bound $UB(T) = f(T.P)$. Once computed, these maximal regions are inserted in Q (Line 17). Notice that this point differs from the processing in BRS, where the algorithm always maintains a single maximal region, which is the best region encountered so far, and eventually the last "surviving" region is the output of the algorithm. Instead, we insert all found maximal regions in Q (including subsequent non-maximal ones) to be further processed in future steps.

The second action is essentially what allows our algorithm to explore the search space around local maxima, thus being able to correctly and progressively retrieve the top-k results for any value of k. Once a slab S has been scanned, in addition to inserting in Q the obtained maximal regions, we also create at most two new slabs, that correspond to the children of S in its slab tree. Each child will inherit the slab sub-tree rooted at itself. The algorithm then computes an upper bound for the children slabs, and inserts them in Q for future processing (Lines 18 and 19). In this way, the top-k BRS algorithm starts examining maximal slabs, and when necessary "backtracks" to examine non-maximal slabs, and generate additional candidate results in a *lazy* and *progressive* manner.

The GetChildren procedure generates at most two slabs S_{back} and S_{fore} from a slab S. Specifically, S_{back} is created when $S.\text{before} > 0$ and gets as its list of points all points in S except the last one. Conversely, S_{fore} is created when $S.\text{after} > 0$ and gets as its list of points all points in S except the first one.

Processing a Region. Finally, when a region T is extracted from Q, it is processed in a similar manner as a slab. Again, there are two actions involved. The first is to treat the region itself as a candidate result. That is, the rectangle R of size $\varepsilon \times \varepsilon$ centered at the center of this region is generated (Line 21). The objective score of R is equal to that of T since, by construction, they contain the same set of points. Thus, R is inserted in the top-k results (Line 22). The second action involves generating the children regions $\mathcal{T}'$ from T, applying the same mechanism as described above for slabs. These are also inserted back to Q to be visited in the future (Lines 23 and 24).

Correctness. The algorithm terminates correctly once k results have been found or once there are no more elements left in Q.

LEMMA 1. *Algorithm* k-BRS *correctly identifies the top-k regions ranked by their objective score.*

PROOF. To establish the correctness of the algorithm, we need to establish the following two points: (i) that the search conducted by the algorithm is complete, i.e., it cannot miss any candidate solutions, and (ii) that the termination condition is correct.

The first is guaranteed by the mechanism for generating derived slabs and regions starting from maximal slabs and maximal regions, respectively. Because all slabs/regions are contained in the slab/region trees associated with the maximal slabs/regions, all candidate rectangles will eventually be produced for examination.

The second point is established by the correctness of the upper bounds computed for the elements in the priority queue. Any rectangle resulting from an element (i.e., a cell, slab or region) in the queue will contain a subset of the points associated with that element. Thus, given that the objective score function f is a monotone function, the objective score of that rectangle cannot be higher than the computed upper bound for the respective element. Consequently, when a result is found, during an iteration of the algorithm, it is guaranteed that none of the elements remaining in the priority queue can produce another result with higher score than it. Hence, it is safe to insert this result in the top-k list, and to terminate the search once k results have been retrieved. □

5 ALGORITHM FOR k-BRS-MG

The important characteristic of the k-BRS algorithm is that it can progressively return the next best result. This allows for a natural mechanism to reevaluate the utility of the next result in the context of the already found ones. We explain this in the following, showing how it applies to the k-BRS-MG problem under the NO and PO variants.

NO variant. In the case of non-overlapping results, this can be accommodated in a rather straightforward way by introducing an additional check at the point where a result is returned. Specifically, recall that when a region T is dequeued from the priority queue Q, it is inserted in the top-k list of results. Instead, in this mode of operation, the algorithm first checks whether the rectangle R corresponding to this region T is overlapping with any of the currently existing rectangles in the top-k list $\mathcal{L}$. If so, then T is discarded, otherwise it is inserted in $\mathcal{L}$. Thus, as before, upon polling a region from Q, a decision can be made immediately about it. The difference is that now the decision may be to discard this candidate result instead of always adding it in the top-k results.

PO variant. The case of partial overlap can be handled with a similar rationale, with certain adaptations described next. Again, when a region is extracted from Q, the corresponding rectangle needs to be evaluated first, in order to check whether or not it qualifies as the next top-k result. This check now involves computing the maximum degree of overlap between this rectangle and any other in the current top-k list, and then using Equations 2 and 3 to compute the marginal gain of this candidate result with respect to the existing ones. The outcomes of this check are now as follows. If the marginal gain is equal to the score of this element according to which it was extracted from Q, it means that this is indeed the next best result, and thus it can be inserted in the top-k list. Otherwise, if this check resulted in a decrease of the element's score, it is pushed back in Q to await for possibly further examination in the future.

Note that it is possible to avoid repeated calculations of the marginal gain of a candidate result as follows. For each visited region, we maintain a counter that indicates the number of results in the top-k list at the last time the marginal gain of this region was computed. Thus, upon extracting a region T from Q, we first check this counter. If it is equal to the current size of $\mathcal{L}$, it means that no other results have been found in the meantime, hence T is the next result to add in $\mathcal{L}$. Otherwise, the aforementioned check needs to be performed to determine whether to keep this result or push it back to Q. Notice that as additional results are being added in $\mathcal{L}$, the marginal gain of any candidate result can never increase. That is, the upper bound of each element in Q continues to be a valid upper bound, as Lemma 2 shows. Thus, once a rectangle is found having a marginal gain higher than the next top element in Q, it is safe to add it to the result set.

The following two lemmas hold for both variants and prove the correctness of the k-BRS-MG algorithm.

LEMMA 2. *Given a list $\mathcal{L}$ of rectangles, the upper bound $UB(X)$ of a queue element X (cell, slab, or region) computed as in Section 4 is an upper bound to the marginal gain of any rectangle containing a subset of points from $X.P$.*

PROOF. From Section 4, we know that the upper bound $UB(X)$ of a queue element X is an upper bound of the score of any rectangle R containing a subset of points from $X.P$. Now, observe that given a list $\mathcal{L}$ of rectangles, the score of a rectangle $R \notin \mathcal{L}$ is an upper bound to its marginal gain, i.e., $f(R.P) \geq mg(R;\mathcal{L})$. This is because the novelty $\gamma(R;\mathcal{L})$ is at most 1 in the NO and PO (for any $\lambda > 0$) modes. Thus, combining the two results, we get that $UB(X)$ is also an upper bound to the marginal gain of R. □

LEMMA 3. *The algorithm correctly finds the solution to the k-BRS-MG problem.*

PROOF. It suffices to show that at the n-th step of the algorithm, i.e., when the top-$(n-1)$ regions are found, the algorithm correctly identifies the next best region. This step terminates when a region with up-to-date marginal gain is extracted from the queue. At that point, Lemma 2 guarantees that all other queue elements cannot contain a candidate result with greater marginal gain.

The last thing we need to show is that any non-seen region, i.e., that has never appeared in the queue, cannot have been the next best region. There are three cases for a non-seen region. First, a non-seen region whose slab was seen, means that there was always in the queue another region of the same slab that had better marginal score than it. This holds because of the way regions within a slab are examined. Second, a non-seen region whose parent slab was not seen and whose parent cell was dequeued, means that there was always in Q another sibling slab that had a better upper bound. This again holds because of the way slabs within a cell are examined. Third, a non-seen region whose parent cell was not dequeued, means that there was always in Q a cell that had a better upper bound. This holds as all cells enter in the queue. □

A final remark concerns an additional optimization for the PO mode. The idea is to lazily compute tighter upper bounds for cells and slabs. Specifically, once a cell or slab is extracted from Q, the upper bound of its marginal gain is lazily updated according to Lemma 4. If the new upper bound is lower than before, it is pushed back to Q instead of being processed. Again here we can avoid repeated calculations by maintaining a counter indicating the size of the top-k list when the previous marginal gain was computed, so that the update only needs to take place if new results have been found in the meantime.

LEMMA 4. *Given a list $\mathcal{L}$ of rectangles, and an element (cell or slab) X, an upper bound to the marginal gain of any rectangle that contains a subset of points from X is given by:*

$$UB(X;\mathcal{L}) = f(S.P) \times e^{-\lambda \cdot \Delta(X;\mathcal{L})}, \text{ where}$$

$$\Delta(X;\mathcal{L}) = \max_{R \in \mathcal{L}} \delta(X,R),$$

$$\delta(X,R) = 1 - \frac{\min\left(area\_size(X.ext \setminus R), area\_size(R)\right)}{area\_size(R)},$$

and $X.ext$ denotes the rectangle produced by expanding the borders of X by $\varepsilon/2$.

PROOF. Any rectangle centered within X is guaranteed to be fully contained within $X.ext$. Consider a rectangle R within the list $\mathcal{L}$. Observe that the size of the area $X.ext \setminus R$ indicates how much free space exists in $X.ext$ that is not overlapping with R. Hence, any rectangle fully contained within $X.ext$, will have at most area_size($X.ext \setminus R$) overlap with R and also certainly not more than its own area size area_size(R).

Therefore, $\delta(X,R)$ is a lower bound $\delta_{min}(X,R)$ on the degree of overlap between any rectangle within X and the rectangle R. A lower bound on the maximum degree of overlap between any rectangle within X and the list $\mathcal{L}$ is then computed by $\Delta(X; \mathcal{L})$, which takes the maximum of the previous value across any rectangle in $\mathcal{L}$. Plugging $\Delta(X; \mathcal{L})$ into function γ gives us an upper bound to the marginal gain of any rectangle within X. □

6 EXPERIMENTAL EVALUATION

In the following, we present an experimental evaluation of our approach, examining the behavior of our algorithms with different datasets. Specifically, we conducted experiments using three real-world datasets containing Points of Interest (POIs) from OpenStreetMap, covering a large number of different categories, such as entertainment, commerce, health, transport and tourism. The three datasets used correspond to the metropolitan areas of Berlin, London and Paris. They contain 53,506, 85,187, and 118,985 POIs, respectively. All algorithms were implemented in Java, and the experiments were conducted on a server with Intel Xeon E5-2420 v2 CPU with 2.20 GHz processor and 64GB RAM running Ubuntu.

6.1 Comparison Based on Marginal Gain

The first experiment aims at providing insight about the quality of the results returned by each ranking method in terms of their novelty. Recall that our motivation is to present to the user a ranked list of rectangles so that (i) their objective score measured by a function f over their enclosed points is maximized, and (ii) the overlap among these rectangles is minimized to avoid repetition and redundancy in the information shown to the user. In our model, the tolerance of a user to this repetition is controlled by the decay constant λ (see Equation 2). The higher the value of λ, the more dissatisfied the user would be by receiving overlapping results.

We conduct this experiment as follows. We assume that the user has a preference indicated by the selected value of λ, and the system supports three modes of operation for ranking the results: (i) *Allow Overlap* (AO) selects and ranks candidate results based only on their objective score, disregarding the criterion of overlap; (ii) *No Overlap* (NO) ranks candidate solutions according to their objective score but excludes any results that overlap with previously selected ones (i.e., it uses the marginal gain resulting from the function $\gamma_{\text{NO}}(\mathcal{L}, R)$ defined in Equation 1); (iii) *Partial Overlap* (PO) ranks results using the marginal gain with exponential decay, i.e., the function $\gamma_{\text{PO}}(\mathcal{L}, R)$ defined in Equation 2. For the PO mode, the user can set the parameter λ to the preferred value, and retrieve results accordingly. In contrast, for the modes AO and NO, the user has no other control over the results, besides this boolean choice. Thus, we then measure the marginal gain of each result in the top-k list for a user having the specified preference for λ.

For this experiment, we set the default value of parameter k to 10. Also, the objective score of each result is normalized in the interval $[0, 1]$ by dividing it with the objective score of the top-1 result. Notice that the top-1 result is the same in all aforementioned modes of operation, AO, NO and PO — specifically, it is the result with the maximum objective score. We set the region width and height ε to 0.001° ($\approx$ 100 meters). This specifies the size of the rectangles to be identified and hence controls also the resolution of the constructed grid. Finally, we vary λ to the values 0.3, 0.4, 0.5.

The results of this experiment are shown in Figure 4. When the ranking algorithm operates in the AO mode, we can always observe a very sharp decrease of the marginal gain already in the top-2 result, which implies that the next retrieved result is highly overlapping with the previous one. Then, the marginal gain remains at these levels for all subsequent results, with little variation. A noticeable case is the top-4 result in Paris, which has a very high marginal gain, compared to that of the top-1 result. In fact, what has happened in this case is that, although the second and third results had a high overlap with the first one, and thus a reduced marginal gain, in the fourth position the algorithm returned a result with similarly high objective score but located in a different area of the city, thus not overlapping with the previously seen ones. Still, the subsequent results were also lying in either of these two areas, thus again having a high degree of overlap and accordingly a smaller marginal gain. Something similar, but to a smaller extent, can be noticed with the result at rank 9.

Overall, for all values of λ, the marginal gain of the top-k results returned by AO deteriorates quickly. To make matters worse, although it is possible to find interesting results further down the list (such as the case of the top-4 result mentioned above), there is no way of knowing if and when a more interesting result will suddenly come along. This means that, for a user who wishes to retrieve and examine results progressively, it is unclear at which point to end the process.

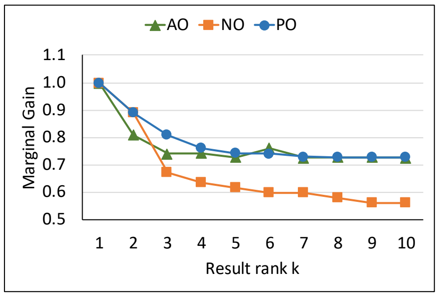
In contrast, the ranking achieved by the modes NO and PO yields results with much higher marginal gain. The difference is smaller in the Berlin dataset, which probably suggests that there is a single or just a few locations having a high concentration of POIs, thus making it more difficult to find diverse results that still have a high objective score, whereas in the London and Paris datasets there may exist a larger number of different areas offering results with high objective score.

Overall, we observe that the marginal gain of both NO and PO decreases at a much lower rate compared to that of AO. Nevertheless, after some point, depending on the selected value for the decay constant λ, the marginal gain of the results returned by NO drops below that of AO. The reason is the following. Being restricted to only return non-overlapping rectangles, NO is inevitably led, after some point, to select rectangles having lower objective scores. Contrary, PO can retrieve results with higher marginal gain at all positions in the list, since it is flexible in balancing the two factors of high objective score and low overlap. Thus, at the beginning it is able to favor non-overlapping rectangles that have comparable objective score to overlapping ones (similarly to NO and conversely to AO). Once the objective score of non-overlapping rectangles starts to drop significantly, it can still admit overlapping ones if their objective score is high enough to compensate for the overlap.

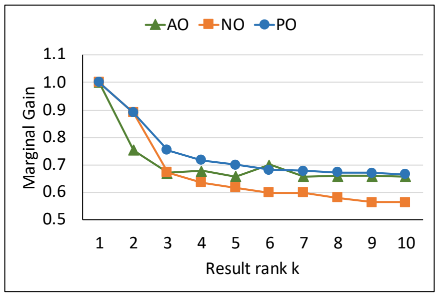
In conclusion, this set of experiments highlights the flexibility offered by PO in returning novel top-k best regions, in contrast to the rigidity of AO and NO.

6.2 Comparison Based on Execution Time

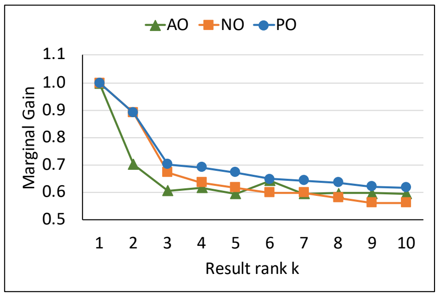
Next, we examine the execution time for each of the three ranking modes, AO, NO and PO, with respect to the parameters k and ε. Moreover, for comparison, we use as baseline an exhaustive search



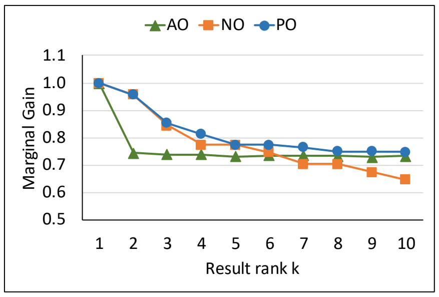
(a) Berlin – $\lambda = 0.3$



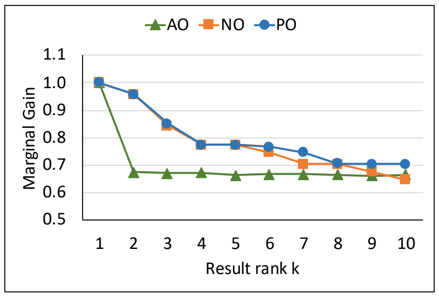
(b) Berlin – $\lambda = 0.4$



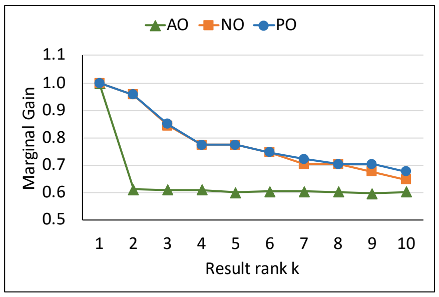
(c) Berlin – $\lambda = 0.5$



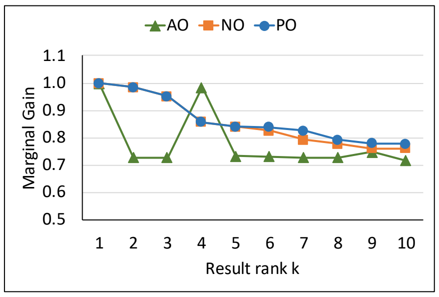
(d) London – $\lambda = 0.3$



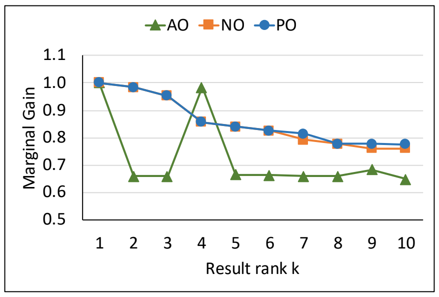
(e) London – $\lambda = 0.4$



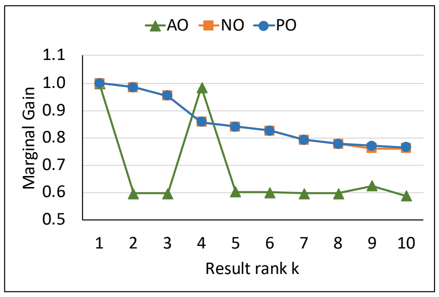
(f) London – $\lambda = 0.5$



(g) Paris – $\lambda = 0.3$

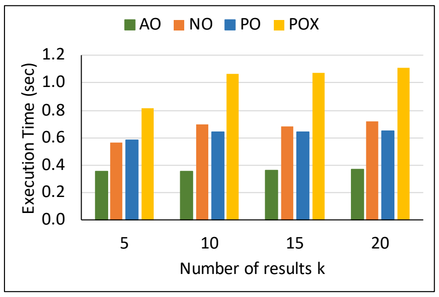


(h) Paris – $\lambda = 0.4$

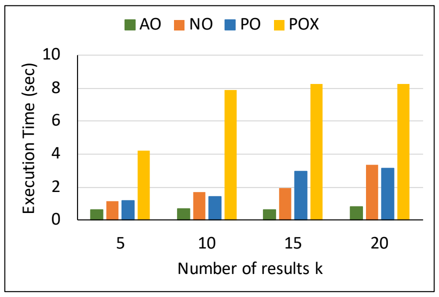


(i) Paris – $\lambda = 0.5$

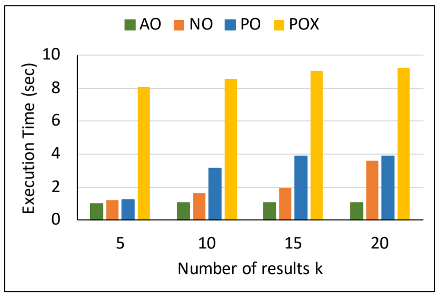
Figure 4: Marginal gain of the top-k retrieved catchment areas by each method.



(a) Berlin



(b) London

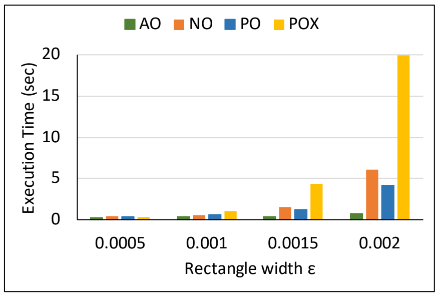


(c) Paris

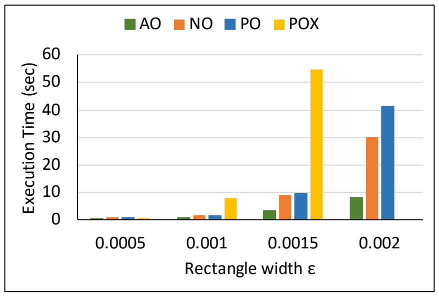
Figure 5: Execution time varying k.

method for the PO mode, which we refer to as POX. Note that POX returns the same top-k list, and generally follows the same hierarchical scan process with PO. The difference lies in that POX does not progressively scan the slabs, exploiting their relationships as captured by the DAG (e.g., see Figure 3). Instead, while executing a sweep, it generates and adds to the queue *all* encountered slabs or regions. This guarantees the completeness of the algorithm, i.e., that the search space of candidate solutions will eventually be fully explored and no results will be missed or produced in wrong order. Nevertheless, as we will see next in the experiments, it fails to do so efficiently, thus allowing to appreciate the advantages of the optimized operation of PO.

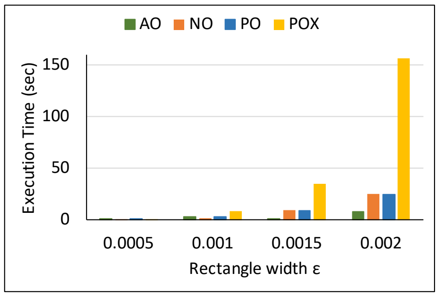
In the conducted experiments, we use $\varepsilon = 0.001°$ as the default value when varying k, and set $k = 10$ when varying ε. For PO, we use 0.4 as the default value for λ. The results are shown in Figures 5 and 6, respectively. Clearly, AO is the fastest method for



(a) Berlin



(b) London



(c) Paris

Figure 6: Execution time varying ε.

retrieving a top-k list of results. This is expected, because in this case the algorithm only needs to search for the next rectangle with the highest objective score. As soon as this is found, it is added to the list without requiring any additional checks that could potentially disqualify it or diminish its marginal gain. Moreover, retrieving subsequent results is relatively fast. Once the grid has been constructed, the priority queue has been initialized, and the top few cells with the highest upper bound have been scanned to identify the top-1 result, then there is a readily available pool of next candidate results to draw from. Nevertheless, AO still suffers from an increase in execution time when ε increases, although to a lower extent compared to NO and PO.

Regarding the last two, both have increased execution times compared to AO. This is expected, since a significantly larger number of candidate results needs to be examined. Specifically, NO needs to continue searching for the next best rectangle until a non-overlapping one is found, whereas PO needs to reinsert a candidate rectangle pulled from the queue back to it if it turns out that its marginal gain is now decreased due to other rectangles having been selected in the meantime. Overall, the two methods have comparable performance. Thus, given that PO allows to tune the preference between objective score and repetition in the retrieved results, while NO does not provide this flexibility, these results indicate that choosing PO between the two is preferable.

It is worth noting that the baseline POX exhibits a significantly higher execution time than all other algorithms in all experiments. This is especially apparent when increasing the value of ε (notably, for $\varepsilon = 0.002^\circ$ in the London dataset, the execution of POX took more than 10 minutes and is thus omitted from the plot). This poor performance of POX is attributed to its naive scanning strategy. As explained, during the sweep it eagerly adds all encountered slabs and regions into the priority queue, thus incurring a significant overhead both in terms of memory use and processing time. Instead, the scanning strategy employed by PO efficiently organizes the maximal slabs/regions and their derived ones, allowing to explore the search space more flexibly and on demand, while still allowing for progressive retrieval of the top-k results.

7 CONCLUSIONS

In this work, we studied practical variants of the *Best Region Search* problem, which seeks the best location for a fixed-size rectangle over a set of geospatial objects so as to maximize a user-defined utility function over its contents. Specifically, we introduced the top-k version where additional results are requested and computed progressively. To overcome repetition and redundancy in the top-k results returned to the user, we proposed diversified variants of the ranking criterion that take into account the degree of overlap among the retrieved results in a way that can be tuned according to the user's preferences for novelty. Our proposed algorithms were shown to be efficient when evaluated on real-world datasets.

ACKNOWLEDGMENTS

This work was partially supported by the EU H2020 project SLIPO (731581) and the NSRF 2014-2020 project HELIX (5002781).

REFERENCES

[1] D. Amagata and T. Hara. A general framework for MaxRS and MaxCRS monitoring in spatial data streams. *ACM Trans. Spatial Algorithms and Systems*, 3(1):1:1–1:34, 2017.
[2] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *PVLDB*, 7(9):733–744, 2014.
[3] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336, 1998.
[4] D. Choi, C. Chung, and Y. Tao. A scalable algorithm for maximizing range sum in spatial databases. *PVLDB*, 5(11):1088–1099, 2012.
[5] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.
[6] K. Feng, G. Cong, S. S. Bhowmick, W. Peng, and C. Miao. Towards best region search for data exploration. In *SIGMOD*, pages 1055–1070, 2016.
[7] K. Feng, T. Guo, G. Cong, S. S. Bhowmick, and S. Ma. SURGE: continuous detection of bursty regions over a stream of spatial objects. *CoRR*, abs/1709.09287, 2017.
[8] K. Feng, T. Guo, G. Cong, S. S. Bhowmick, and S. Ma. SURGE: Continuous detection of bursty regions over a stream of spatial objects. In *ICDE*, 2018.
[9] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.
[10] H. Imai and T. Asano. 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] J. Liu, G. Yu, and H. Sun. Subject-oriented top-k hot region queries in spatial dataset. In *CIKM*, pages 2409–2412, 2011.
[12] P. Mehta, D. Skoutas, D. Sacharidis, and A. Voisard. Coverage and diversity aware top-k query for spatio-temporal posts. 2016.
[13] 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(8):45–61, 1995.
[14] D. Sacharidis, P. Mehta, D. Skoutas, K. Patroumpas, and A. Voisard. Continuous summarization of streaming spatio-textual posts. 2017.
[15] Y. Tao, X. Hu, D. Choi, and C. Chung. Approximate MaxRS in spatial databases. *PVLDB*, 6(13):1546–1557, 2013.
[16] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *ICDE*, pages 1163–1174, 2011.
[17] R. C. Wong, M. T. Özsu, P. S. Yu, A. W. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.
[18] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *ICDE*, pages 804–815, 2011.