Discovering Mixture-Based Best Regions of Arbitrary Shapes

Dimitrios Skoutas
Athena Research Center
Athens, Greece
dskoutas@athenarc.gr

Dimitris Sacharidis
Université Libre de Bruxelles
Brussels, Belgium
dimitris.sacharidis@ulb.be

Kostas Patroumpas
Athena Research Center
Athens, Greece
kpatro@athenarc.gr

ABSTRACT

Given a collection of geospatial points of different types, mixture-based best region search aims at discovering spatial regions exhibiting either very high or very low mixture with respect to the types of enclosed points. Existing works detect fixed-shape regions, such as circles or rectangles, thus often missing interesting regions occurring in real-world data that may have arbitrary shapes. In this paper, we formulate the problem of mixture-based best region search for arbitrarily shaped regions, introducing certain desired properties to ensure their cohesiveness and completeness. Since computing exact solutions to this problem has exponential cost with respect to the number of points, we propose anytime algorithms that efficiently search the space of candidate solutions to produce high-scoring regions under any given time budget. Our experiments on several real-world datasets show that our algorithms can produce high-quality results even within tight time constraints.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Geographic information systems**.

KEYWORDS

spatial mixture pattern, areas of interest, spatial analytics

ACM Reference Format:
Dimitrios Skoutas, Dimitris Sacharidis, and Kostas Patroumpas. 2021. Discovering Mixture-Based Best Regions of Arbitrary Shapes. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21), November 2–5, 2021, Beijing, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3474717.3484215

1 INTRODUCTION

The growing amount of geospatial data offers a wealth of information about locations and human activities [10]. In particular, Points of Interest (POIs) cover a broad variety of geolocated entities, such as businesses, public services, tourist attractions, constituting the cornerstone of many applications in the domains of geomarketing, navigation, logistics, mobile advertisement, urban planning and others [4]. Analyzing and mining POI data can reveal important insights about regions of particular interest within a study area.

Detecting *areas of interest* refers to finding regions that exhibit some particular property. Examples include detecting spatial hotspots using global and local spatial autocorrelation [3] or density-based clustering [11]. Other methods combine spatial distance with keyword search to find groups of POIs that are located close to each other and collectively cover a set of user-defined keywords [5, 6, 8, 9]. In the Maximizing Range Sum (MaxRS) problem [7, 12, 15–17], the goal is to find a rectangular region of user-specified width and height that maximizes the number or total weight of enclosed points. Other methods have focused on detecting regions that exhibit interesting spatial mixture patterns [14, 18].

In this paper, we address the latter problem, aiming to detect regions characterized by high or low diversity with respect to the types of points they contain. Spatial mixture patterns provide useful insights for decision making in various applications. For instance, *high mixture* regions may be more attractive to customers seeking to purchase or rent a new residence, as they offer a larger variety of amenities. On the other hand, *low mixture* patterns reveal areas that emphasize on specific activities, such as a business district.

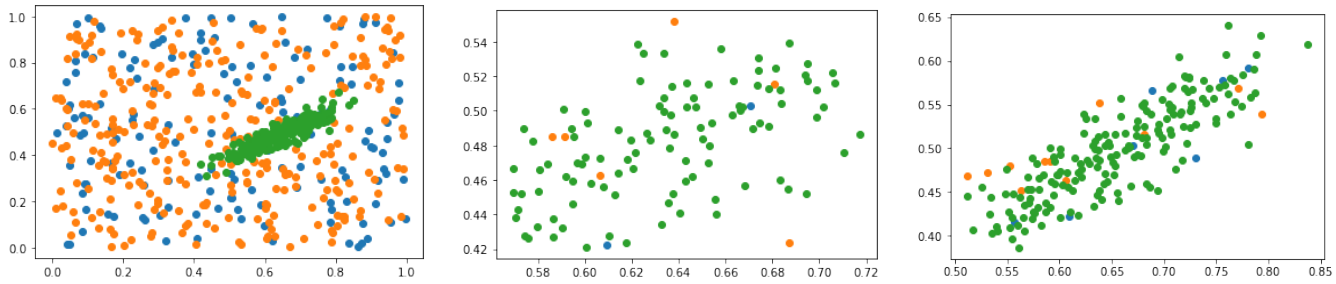
Existing approaches scan the area of study by exhaustively enumerating candidate regions of a certain shape, i.e., circle or square [14, 18]. However, such fixed-shape regions may not be well-aligned with the actual shapes of high or low mixture regions occurring in the real world. In practice, such regions often exhibit *arbitrary shapes* imposed by various natural barriers (e.g., lakes, rivers) or man-made structures (e.g., pedestrian streets).

Figure 1(a) shows a motivating example using a synthetic dataset comprising three different types of points (depicted with different colors). Blue and orange points are uniformly distributed, whereas green points are clustered. This resembles common situations in real-world data, e.g., in the case of a commercial street with many shops along each side. Assume that we are interested in low mixture regions. The algorithm should detect the region corresponding to the cluster of green points. Figure 1(b) shows the region detected by an algorithm that scans the area using a regular geometric shape (e.g., circle as detailed in Section 4.1). The result includes only a small portion of the desired region. The reason is that if the circle is enlarged further, to include a larger portion of the green points, this will also introduce many blue and orange points, thus producing a region that no longer exhibits low mixture. In contrast, as shown in Figure 1(c), our algorithm that detects arbitrarily shaped regions (see Sections 4.2 and 4.3) is able to discover the desired region.

However, for regions of arbitrary shapes, it is challenging how to formulate the problem and to design efficient and effective algorithms for detecting them. To this end, we propose a graph-based approach and design several anytime algorithms employing alternative strategies to efficiently and effectively scan the search space. Given a set of POIs, each one belonging to a certain category, our method first constructs a spatial connectivity graph, where edges

SIGSPATIAL '21, November 2–5, 2021, Beijing, China

ACM ISBN 978-1-4503-8664-7/21/11…$15.00
https://doi.org/10.1145/3474717.3484215



(a) Input containing points of three different types.

(b) Result produced by algorithm detecting fixed-shape (circular) regions.

(c) Result produced by algorithm detecting arbitrarily shaped regions.

Figure 1: Motivating example illustrating the detection of a low-mixture region on a synthetic dataset.

exist between pairs of POIs that are within a user-defined distance threshold from each other. A region of interest is then defined as a subgraph in this graph. To obtain valid (i.e., meaningful) regions, we introduce a set of constraints that ensure the cohesiveness and completeness of each region. We also define the interestingness score of a region based on the entropy of a feature vector representing the distribution of the POI types it encloses, as well as its size. The latter is used to favor larger regions over smaller ones, since patterns involving only a few points may also occur by chance.

When allowing arbitrary shapes, and given that the entropy score of a region is not monotonic with respect to its size, computing exact solutions requires enumerating an exponential number of subgraphs, which is infeasible in practice. To overcome this, we design greedy, *anytime* algorithms, that try to detect the best solution under any given time budget. As a baseline, we enhance a method for fixed shape regions with post-processing steps to produce valid regions under our graph-based formulation. Then, we describe an algorithm that enumerates subgraphs according to certain expansion criteria and uses a priority queue to explore the search space of candidate regions under the given time budget. We discuss the impact of the number of seeds used for subgraph expansion, as well as the resolution of the expansion criteria, on the time versus quality tradeoff for the detected regions. These observations lead us to design more effective search strategies that better balance the exploration and exploitation of candidates, thus optimizing the utilization of the available time budget.

Our experiments show that the added flexibility of arbitrary shapes allows to discover better regions compared to those found by methods that only detect fixed-shape regions. Moreover, our methods for prioritizing the enumeration and evaluation of candidates decrease the time to discover high-scoring regions, leading to anytime algorithms that are more suitable for data exploration.

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first to formulate and study the problem of mixture-based best region search for arbitrarily shaped regions.
- We adapt an existing algorithm from spatial scan statistics to detect valid regions under our graph-based formulation.
- We introduce an algorithm based on subgraph expansion to enumerate and prioritize the generation and evaluation of arbitrarily shaped candidate regions.
- We extend our approach with additional search strategies that more effectively prioritize the exploration of the search space of candidate regions, thus leading to more optimal utilization of the available time budget.
- We conduct an extensive experimental evaluation using several real-world POI datasets to compare the proposed algorithms and study the effects of the involved parameters.

The remainder of the paper is structured as follows. Section 2 reviews related work. Section 3 formally defines the problem. Section 4 presents the proposed algorithms. Section 5 discusses our experimental evaluation, and Section 6 concludes the paper.

2 RELATED WORK

Several methods exist for detecting *areas of interest* under various criteria. In the following, we review different lines of research.

Spatial mixture patterns. The closest work to our setting is a recently proposed approach for detecting spatial mixture patterns [18]. It identifies regions having significantly high or low mixture of point types, overcoming two main issues in previous studies. The first is natural randomness yielding spurious results that qualify by sheer chance. The second refers to inherent mixture patterns due to increased variety in point categories. Unlike multinomial scan statistic methods like [14] that detect sub-regions with different mixture patterns than their surrounding area, it introduces a Spatial Mixture Index (SMI) to rank and select candidate regions. The proposed algorithm enumerates candidate circular regions around uniformly sampled seed points, computes their score according to the SMI, and picks the one with the maximum score for significance testing. Evaluation also involves a dual-level Monte-Carlo estimation: first on candidate-level around same-sized regions around the chosen seeds, and then on data-level to get the maximum score achieved in each random subset of the data. However, the basic constraint of this method is that returned regions are bound by a circular shape, as all candidate regions are generated with incremental radii around the initial seeds.

Spatial clusters. Spatial hotspots can be detecting using measures of global and local spatial autocorrelation [3]. Density-based clustering methods like DBSCAN [11] measure density using distances between data points to identify clusters of various shapes and separate them from noise. Integrating statistical significance to DBSCAN was suggested in [19] to eliminate candidate clusters

that are most likely formed by chance, while still finding clusters of varying shapes and densities. A recent survey [20] offers a taxonomy and in-depth review of models and algorithms for statistically robust clustering. Although density-based clustering can detect arbitrarily shaped regions, it is not straightforward how to apply such algorithms for detecting high or low mixture patterns.

Maximizing Range Sum. The *Maximizing Range Sum* (MaxRS) problem [7] finds the optimal placement of a fixed-size rectangle that maximizes the total weight of points therein. An $(1-\epsilon)$-approximation was proposed in [17]. For *continuous MaxRS* monitoring in streams, a branch-and-bound approximation algorithm with worst-error guarantees is proposed in [2]. *Best Region Search* (BRS) [12] generalizes MaxRS, allowing any submodular monotone function for score estimates. The algorithm considers each point as the center of a fixed-size rectangle and identifies the regions that are maximal intersections of these rectangles. The k-BRS problem [16] progressively returns the next best result to compile a ranked list of the top-k best rectangles. Diversification is also employed to avoid overlapping regions in the top-k results. Parallel and distributed algorithms for k-BRS have been proposed in [15].

Spatial-keyword search. These approaches combine spatial distance with keyword search to find groups of closely located POIs that collectively cover a set of user-defined keywords. The length-constrained maximum-sum region (LCMSR) problem [6] finds a spatial-network region located within a broader area of user's interest, containing up to a maximum number of POIs that best match query keywords, and also maximizing the total weight of such POIs with respect to the keywords. Given a query location and a set of keywords, the algorithms in [5] retrieve groups of spatio-textual objects that collectively meet user's preferences, but the returned objects are also close to each other to indicate an area of interest. Similarly, the spatial keyword cover problem (SK-COVER) [8] finds the group of objects that covers all query keywords and minimizes a distance cost function; as a result, fewer collocated objects are contained in the returned group. Besides, the Best Keyword Cover problem [9] finds objects that apart from covering the query keywords and minimum distance among themselves, also includes the keyword rating during evaluation. In all these cases, the user must specify a query with various criteria (e.g., location, keywords, maximum number of POIs). Instead, we treat our problem as a data discovery problem without specifying query locations or categories.

3 PROBLEM DEFINITION

In this section, we introduce the basic concepts and notations used throughout the paper, and we formally define the problem of mixture-based best region search. Basic notations are in Table 1.

Assume a collection $\mathcal{D}$ of points in a 2-dimensional space with each point $p \in \mathcal{D}$ belonging to a certain class $\tau \in \mathcal{T}$ from a given set of classes $\mathcal{T}$. We represent each point as a tuple $p = (x, y, \tau)$, where (x, y) denotes its coordinates and τ denotes its type.

A region R is a subset of points in $\mathcal{D}$. Before we introduce the properties that constitute a *valid* region for our problem, we first discuss how to assign a score to a given region R that quantifies how interesting it is from the point of view of the diversity of the types of its enclosed points.

Table 1: Notations

$\mathcal{T}$	Set of categories used in point classification
$\mathcal{D}$	Point dataset; each point classified to a category from $\mathcal{T}$
M	Maximum region size (count of points)
ϵ	Distance threshold
G	Spatial Connectivity Graph w.r.t. distance threshold ϵ
S	Set of seed points chosen from $\mathcal{D}$
R^*	Best region consisting of at most M points
T	Time budget allocated to find the best possible region
N_v	Neighborhood of points w.r.t. point v
$\sigma(R)$	Interestingness score assigned to region R
γ	Weight of the size factor in score computations

Region score. Let $\phi : R \rightarrow [0, 1]$ be a function that assigns a score to a given region R based on the type of points it contains. We are interested in regions that have high values of ϕ, while also preferring larger regions compared to smaller ones, since mixture patterns involving just a few points may occur merely by chance. To combine these two criteria, we define the interestingness score of a region R as follows:

Definition 1 (Region Score). *The interestingness score of a region R is defined as:*

$$\sigma(R) = \phi(R) \cdot (|R|/M)^{\gamma} \tag{1}$$

where $\phi(R) \in [0, 1]$ is a score computed based on the mixture of types of points in R, M is the maximum allowed size of a region, and γ is a parameter that determines the weight of the size factor. Notice that $\sigma(R) \in [0, 1]$.

Similarly to [18], we use Shannon's entropy to measure the diversity of types within a region; however, other measures can be applied as well. Given the types of points in a region R, Shannon's entropy is defined as follows:

$$H(R) = -\sum_{\tau \in \mathcal{T}} \rho_\tau(R) \cdot \log \rho_\tau(R) \tag{2}$$

where ρ_τ is the portion of points in R belonging to type τ, i.e.,

$$\rho_\tau(R) = \frac{|\{p : p \in R \;\&\; p.\tau = \tau\}|}{|R|} \tag{3}$$

We can now define function ϕ based on the above. In particular, we introduce two alternative variants, ϕ_H and ϕ_L, depending on whether we are interested in regions having a high or low mixture, respectively:

$$\phi_H(R) = H(R)/H_{max} \tag{4}$$

$$\phi_L(R) = 1 - H(R)/H_{max} \tag{5}$$

where $H_{max} = \log |\mathcal{T}|$ is the maximum value of Shannon's entropy for a set of classes $\mathcal{T}$.

Valid regions. Our goal is to discover interesting regions that have *arbitrary shapes*, as opposed to existing methods that rely on a fixed shape (typically, circle or rectangle) to define a region. Since now the shape of the region cannot be used to define candidate regions, we need to introduce other criteria to ensure that discovered

regions are meaningful. To that end, we introduce two properties, *cohesiveness* and *completeness*, which a valid region must satisfy.

Cohesiveness ensures that the points forming a valid region are spatially close to each other. More specifically, each point should be within a user-defined distance threshold ϵ from at least one other point in the region. In the case of regular-shaped regions, this property is to some extent implied by the fact that all points in the region are enclosed within the chosen fixed shape (e.g., circle or rectangle); nevertheless, this does not necessarily prevent the case that certain points are relatively far away from others (i.e., having much "dead space" within the circle or rectangle).

Completeness ensures that any point in the dataset that is spatially close (i.e., within distance ϵ) to a point in the region is also part of the region, with the exception of the *border* points of the region. Intuitively, this implies that all points enclosed within the border of a region should also be considered as members of that region; otherwise, one could trivially "hand-pick" subsets of points from the study area to form "regions" that artificially exhibit a high or low mixture pattern.

To evaluate both cohesiveness and completeness when searching the space of valid regions, we represent the collection $\mathcal{D}$ with an underlying *spatial connectivity graph* G. Formally:

Definition 2 (Spatial Connectivity Graph). *Given a collection $\mathcal{D}$ of points, a distance function d, and a distance threshold ϵ, the spatial connectivity graph G is a graph $G = (V, E)$, where V is the set of points and E is the set of edges such that $(u, v) \in E$ if $d(u, v) \leq \epsilon$.*

In our setting, we deal with points in 2-dimensional space and we assume Euclidean distance; nevertheless, the spatial connectivity graph is generic and could be applied in different settings as well, e.g., using the connections between points in an underlying road network. Since we only deal with valid regions, in the sequel we refer to valid regions simply as regions.

We can now formally define the concept of a (valid) region as follows.

Definition 3 (Region). *Given a spatial connectivity graph G, a (valid) region R is a subgraph $G_R = (V_{R_C} \cup V_{R_B}, E_R)$ of G, where V_{R_C} and V_{R_B} comprise the* core *and* border *nodes of R, respectively, such that the following conditions are satisfied:*

1. *The set of core points V_{R_C} forms a connected subgraph in G, i.e., each core node $v \in V_{R_C}$ is reachable from any other core node $u \in V_{R_C}$ via a path containing other core nodes.*
2. *Each border point is connected to at least one core point, i.e., for any border node $v \in V_{R_B}$ there exists at least one core node $u \in V_{R_C}$ such that $(v, u) \in E_R$.*
3. *All neighbors of a core point are also part of the region, i.e., for any core node $v \in V_{R_C}$ if u is a neighbor of v (i.e., $(v, u) \in E_R$) then $u \in V_{R_C} \cup V_{R_B}$.*

In the above definition, conditions (1) and (2) are introduced to ensure the cohesiveness of a region, while condition (3) refers to the criterion for completeness.

Problem Statement. We can now formally define the problem addressed in this paper.

Problem 1 (Mixture-Based Best Region Search (M-BRS)). *Given a spatial connectivity graph G representing a collection of points $\mathcal{D}$ belonging to different types $\mathcal{T}$, and a maximum region size M, find the region with the maximum score $\sigma(R)$, i.e.,*

$$R^* = \underset{R \subseteq G, |R| \leq M}{\arg\max} \ \sigma(R) \tag{6}$$

Our approach. A key observation is that, in our problem, the score of a region does not exhibit monotonicity with respect to its size. In other words, whenever a region is expanded with additional points, its score may either increase or decrease, depending on whether the newly introduced points have types that increase or decrease the total entropy of types distribution in the resulting region. For instance, the score of a candidate region may decrease after a single expansion but may then increase again if the new region is expanded even further. This lack of monotonicity prevents the use of algorithms similar to those proposed for best region search [12, 16] or Apriori-like algorithms [1]. On the other hand, exhaustively enumerating all subgraphs of G to identify and evaluate candidate regions is clearly infeasible for large, real-world datasets. Consequently, to address this problem, we focus on designing *anytime* algorithms, i.e., algorithms that aim at detecting regions of as high score as possible under any given time budget T.

4 ALGORITHMS

In this section, we present several anytime algorithms for addressing the mixture-based best region search problem defined above. We start with a baseline algorithm that enumerates candidate regions relying on a fixed geometric shape. Then, we introduce an algorithm that performs subgraph expansion in the spatial connectivity graph to detect arbitrarily shaped regions. Finally, we present how the anytime characteristic of this algorithm can be improved by more effectively exploring the search space.

4.1 Fixed-shape Scan

As a baseline, we consider an algorithm, called CircularScan, which enumerates candidate regions by scanning the study area with a fixed geometric shape, in particular a circle. We enhance this process with necessary adaptations to ensure that valid regions are produced at each step, according to Definition 3.

Candidate enumeration. Candidate enumeration is performed by scanning the area with a regular geometric shape (in our case, we choose a circle). This is typical in spatial scan statistics [13, 14], and is also adopted by the current state-of-the-art method for detecting spatial mixture patterns [18]. According to this method, candidate regions are generated by exhaustively enumerating all circles (up to a maximum size) having a point $p \in \mathcal{D}$ as their center and another point $p' \in \mathcal{D}$ on their circumference. As pointed out in [18], to avoid repeatedly executing multiple range queries for each considered center point p, it is possible to first sort all other points by increasing distance to p and then incrementally expand the circle by adding the next nearest neighbor at each step. Moreover, if the number of points in $\mathcal{D}$ is large, it is possible to select only a subset $S \subseteq \mathcal{D}$ of them as centers, e.g., using uniform sampling.

Adaptations. Next, we enhance the above candidate enumeration method with two adaptations. The first adaptation allows the CircularScan algorithm to run in an anytime fashion, similarly to the other approaches presented next in Sections 4.2 and 4.3. Notice that the basic enumeration method described above visits the

selected center points in a random order and fully enumerates all candidate circles around each one before moving on to the next. This means that it may take a long time until a high-scoring region is detected, thus possibly failing to detect regions of high quality if the given time budget is limited. In the GraphExpand algorithm (presented in Section 4.2), this is addressed by introducing a priority queue that maintains the currently active candidate regions, and prioritizes their expansion based on their current score. Thus, we modify CircularScan to adopt the same technique, i.e., to replace the sequential evaluation of candidates with one determined by a priority queue. Queue Q is initialized with scores of candidate regions consisting of their center and its nearest neighbor from dataset $\mathcal{D}$ for each region. Typically, after a region is examined for expansion, it may be pushed back to Q with its updated score, as long as its current size does not exceed the maximum size M.

The second adaptation involves some additional processing required to obtain a valid region (according to Definition 3) from the set of points enclosed in a candidate circle produced by the enumeration process. Specifically, we need to ensure that regions produced by CircularScan meet the cohesiveness and completeness requirements for valid regions. To satisfy cohesiveness, when enumerating circles around a center point p, we check whether the next neighbor p' lies within distance ϵ from at least one of the points in the current region R formed around p. If not, then point p' is skipped, and does not become part of region R. Moreover, if $d(p, p') - d(p, p_{last}) > \epsilon$, where p_{last} refers to the last point that was used to extend the circumference of the circle, then the circle is not expanded further, since no newly found points can be within distance ϵ from existing points in R. Eventually, the accumulated points constitute the set of core points of R, which has been ensured to form a connected subgraph. Also, to satisfy the completeness property, any neighbor of a core point that is not already part of R is retrieved; these points form the border of R. If R has higher score than the best region R^* found so far at a previous iteration, then R itself becomes the current best region R^*.

4.2 Graph Expansion

In this section, we present our algorithm, called GraphExpand, which is based on *subgraph expansion* in the spatial connectivity graph G to detect the best arbitrarily shaped region R^* of maximum size M under a given time budget T. We first present an overview of the algorithm and then describe its main parts in more detail.

Overview. The main steps of GraphExpand are outlined in Algorithm 1, and comprise two main phases, namely *initialization* and *expansion*. The initialization phase (Lines 1–3) involves selecting a set of seed points S, initializing their respective regions, and using these regions to populate a priority queue Q that is used to prioritize candidate enumeration in the expansion phase. In our implementation, to select the set of seeds S, we select $\rho \cdot |V_G|$ nodes of the spatial connectivity graph G uniformly at random, where $\rho \in (0, 1]$ is a parameter of the algorithm. We examine the impact of ρ, i.e., the number of seeds, in our experiments.

Next, the expansion phase (Lines 4–11) iterates over the candidate regions in the priority queue Q until either there are no more remaining entries in Q or the time budget T is exhausted. Whenever a new entry is retrieved from Q, the region is expanded to generate a new candidate R. If the size of R does not exceed the maximum allowed region size M, then we check whether its score is higher than the score of the currently known best region R^*, and if so, R^* is replaced by R. Also, if the border of R is not empty, R is pushed to Q to be further expanded in a future iteration.

Algorithm 1: GraphExpand

Input: Spatial Connectivity Graph G, Max Region Size M, Time Budget T
Output: Best Region R^*

▷ Phase I: Initialization
1 $R^* \leftarrow \emptyset$ ▷ initialize top region
2 $S \leftarrow pick\_seeds(G)$ ▷ select a subset of V_G as seeds
3 $Q \leftarrow init\_queue(G, S)$ ▷ initialize priority queue
▷ Phase II: Expansion
4 **while** $elapsed\_time < T$ and $|Q| > 0$ **do**
5 $R \leftarrow Q.pop()$ ▷ get the next candidate region
6 $R \leftarrow expand\_region(G, R)$ ▷ expand the region
7 **if** $|R| \leq M$ **then** ▷ check max size
8 **if** $R.score > R^*.score$ **then** ▷ update top region
9 $R^* \leftarrow R$
10 **if** $|V_{R_B}| > 0$ **then** ▷ add to queue if border not empty
11 $Q.push(R, R'.score)$
12 **return** R^*

Algorithm 2: Procedure InitQueue()

Input: Spatial Connectivity Graph G, Set of Seed Nodes S
Output: Priority Queue Q

1 $Q \leftarrow \emptyset$ ▷ priority queue
2 **foreach** $v \in S$ **do**
3 $R \leftarrow create\_region(G, v)$ ▷ create region R with v as core point
4 $R \leftarrow refine\_border(G, R)$ ▷ check if certain border points are core
5 **if** $|R| \leq M$ and $|V_{R_B}| > 0$ **then** ▷ check max size and border
6 $Q.push(R, \sigma)$ ▷ add region to queue
7 **return** Q

Initialization. The algorithm GraphExpand relies on a priority queue to prioritize the evaluation of the enumerated candidate regions. Each generated candidate region, both during the initialization and the expansion phase, is inserted in this queue according to its score. In this way, instead of fully expanding all candidate regions around a seed node before moving on to the next one, the algorithm picks the next candidate region to expand in decreasing order of their current score. Notice that, although this increases the total execution time, due to the added overhead of maintaining the priority queue, it enables the algorithm to run in an anytime manner, i.e., it increases the likelihood of detecting high scoring regions as early as possible.

The process to initialize the priority queue is described in Algorithm 2. Each seed point is visited to create its corresponding valid region (Lines 2–6). Specifically, as detailed in Algorithm 3, given a seed node v, this region comprises v as its core point and all the neighbors of v as its border points. Also, the score of the region is computed according to Equation 1. Notice that the creation of a

Algorithm 3: Procedure CreateRegion()

Input: Spatial Connectivity Graph G, Node v
Output: Initial Region R

1 $V_{R_C} \leftarrow \{v\}$ ▷ add v as core
2 $V_{R_B} \leftarrow \emptyset$ ▷ initialize border
3 $E_R \leftarrow \emptyset$
4 $N_v \leftarrow get\_neighbors(G, v)$ ▷ retrieve the neighbors of v
5 **foreach** $u \in N_v$ **do**
6 $\quad V_{R_B} \leftarrow V_{R_B} \cup \{u\}$ ▷ add u as border point
7 $\quad E_R \leftarrow E_R \cup \{(v, u)\}$ ▷ connect border point u to core point v
8 $R.score \leftarrow compute\_score(R)$ ▷ compute score according to Eq. 1
9 **return** R

Algorithm 4: Procedure ExpandAll()

Input: Spatial Connectivity Graph G, Current Region R
Output: Expanded Region R

1 **foreach** $v \in V_{R_B}$ **do** ▷ expand all border points
2 $\quad R \leftarrow expand\_point(G, R, v)$
3 $R \leftarrow refine\_border(G, R)$ ▷ check if certain border points are core
4 **return** R

Algorithm 5: Procedure ExpandBest()

Input: Spatial Connectivity Graph G, Current Region R
Output: Expanded Region R'

1 $R_{best} \leftarrow R$ ▷ initialize new region
2 **foreach** $v \in V_{R_B}$ **do** ▷ iterate over all border points
3 $\quad R' \leftarrow expand\_point(G, R, v)$ ▷ expand this border point
4 $\quad$ **if** $|R'| \leq M$ and $R'.score > R_{best}.score$ **then**
5 $\quad\quad R_{best} \leftarrow R'$ ▷ update new best region
6 $R_{best} \leftarrow refine\_border(G, R_{best})$ ▷ check if certain border points are core
7 **return** R_{best}

region R is followed by a post-processing step (Line 4) to refine its border (pseudo-code omitted for brevity). This procedure iterates over each border point, retrieves its neighbors, and checks whether all of these neighbors already belong to R. If so, then this point is removed from the set of border points and added to the set of core points. Although this process is not necessary to ensure the correctness of the algorithm, detecting border points that are actually core points and removing them from the border set of the region avoids unnecessary computations during the expansion phase. Finally, if R does not exceed the maximum allowed size M and the region border is not empty, R is inserted to the queue (Lines 5–6).

Expansion. Next, we discuss the expansion process. Assume a current candidate region R that is retrieved from Q. R corresponds to a subgraph in the spatial connectivity graph G. The purpose of the expansion process is to expand R, generating one or more new candidate regions, whose nodes are a superset of R. The expansion works by selecting one or more border points of R to be expanded, according to some selection criterion. These border points become core points, and all their neighbors are retrieved and added to the

Algorithm 6: Procedure ExpandPoint()

Input: Spatial Connectivity Graph G, Current Region R, Border Point v to Expand
Output: Expanded Region R'

1 $R' \leftarrow R$ ▷ initialize new region
2 $V_{R'_B} \leftarrow V_{R'_B} \setminus \{v\}$ ▷ remove v from border
3 $V_{R'_C} \leftarrow V_{R'_C} \cup \{v\}$ ▷ add v to core
4 $N_v \leftarrow get\_neighbors(G, v)$ ▷ retrieve the neighbors of v
5 **foreach** $u \in N_v$ **do**
6 $\quad V_{R'_B} \leftarrow V_{R'_B} \cup \{u\}$ ▷ add u as border point
7 $\quad E'_R \leftarrow E'_R \cup \{(v, u)\}$ ▷ connect border point u to core point v
8 $R'.score \leftarrow compute\_score(R)$ ▷ compute score according to Eq. 1
9 **return** R'

set of border points of the newly formed region. The question is how many and which border points to expand. We consider two selection criteria: (a) select all border points (procedure ExpandAll in Algorithm 4), and (b) select the border point which leads to the highest scoring region once expanded (procedure ExpandBest in Algorithm 5). Notice that both expansion strategies generate exactly one new candidate region out of an initial one (or zero, if no further expansion is possible, due to the maximum size limit M). This choice is preferred as it ensures that the size of the priority queue is non-increasing, i.e., Q only grows during the initialization phase and then at each iteration its size remains the same or decreases by one. It is straightforward to implement several alternative expansion strategies, e.g., to expand each border point and push the new region to the priority queue, or to expand the k-best border points; however, our experimentation with such alternative criteria has shown that these choices incur a significant increase in the size of Q, while also generating candidate regions that are quite similar to each other, thus eventually slowing down rather than speeding up the discovery of highly scoring regions.

Both procedures ExpandAll and ExpandBest internally employ procedure ExpandPoint (see Algorithm 6), which expands a current region R with a given border point v. This is performed by moving v from the set of border points to the set of core points, adding all neighbors of v to the border set of R, and eventually computing the score of the new region. The difference is that ExpandAll expands the entire set of border points, whereas ExpandBest examines each border point individually and eventually selects only one of them to expand (i.e., the one leading to the expanded region with the highest score). Either of these two procedures can be used in Line 6 of Algorithm 1 to expand a given region.

Discussion. As mentioned above, both procedures ExpandAll and ExpandBest receive as input one region and produce as output (at most) one new candidate. Therefore, the size of the priority queue Q is bound by the number $|S|$ of selected seeds during initialization. Moreover, at each expansion, at least one new point is added to the current region; hence, the number of expansions that may occur for each seed is at most M. Finally, during an expansion, each border point of the current region has to be considered. Consequently, the time complexity of algorithm GraphExpand is $O(|S| \cdot M \cdot d)$, where d is the average node degree in G.

In practice, ExpandAll and ExpandBest offer a different trade-off between expansion speed and granularity. In ExpandAll, the entire border is expanded in each iteration, hence the maximum region size M is reached much faster, i.e., the expansion for each given seed will terminate faster. Instead, in ExpandBest, regions grow at a slower pace, but the algorithm can select to expand only favorable border points. Intuitively, assuming that both expansion strategies are given the same time budget, ExpandAll can utilize it to examine a larger set of seed points, whereas ExpandBest can examine fewer seed points but more thoroughly. Regardless of this difference, both strategies suffer from a common drawback, which is the fact that seed selection takes place in advance. These seeds are used to populate the priority queue, which drives the execution of the algorithm from that point onward; hence, adding or modifying the set of seeds dynamically is not straightforward.

4.3 Adaptive Seed Prioritization

As explained above, the main drawback of both ExpandAll and ExpandBest is that they rely on the a priori selection of a fixed set of seed points. Yet, as also confirmed by extensive experiments (indicative ones reported in Section A.2), effectively managing the placement of seeds is crucial when operating under a limited time budget. Failing to choose appropriate seeds for expansion leads the algorithm to detecting suboptimal regions. When the time budget is limited, the problem cannot be trivially tackled by simply increasing the number of selected seeds, since this also means higher execution time. Instead, the number of utilized seeds needs to be maintained at low levels; hence, their allocation needs to be managed more effectively.

The above observations lead us to designing alternative search strategies that are based on selecting seeds dynamically and adaptively during execution of the algorithm. Specifically, in this section, we present two such algorithms, based on the same basic idea: instead of basing the search on *seed points*, we use *seed areas*. More specifically, instead of drawing all seeds at once from the entire study area, we use a few seeds as starting points to gain some initial knowledge about the dataset, and then dynamically adapt the selection of subsequent seeds accordingly. The difference between the two algorithms described next lies mainly on how these seed areas are defined. In the first method, they are candidate regions produced using the ExpandAll strategy around initial seeds, whereas in the second they are the cells of a uniform grid. Nevertheless, other such alternative search strategies can be designed as well, following similar ideas and the same underlying rationale.

Adaptive Hybrid. The first algorithm is called AdaptiveHybrid because it combines ExpandAll and ExpandBest in a two-step process. In the first step, a small number of starting seeds is randomly selected. Each seed is examined sequentially. First, its corresponding region is initialized, using procedure CreateRegion, and then this region is iteratively expanded, using procedure ExpandAll, until the maximum size M is reached or no new neighbor nodes exist. The best region found by each initial seed is marked as a *seed area* and is pushed to a priority queue sorted by its region score. Hence, the result of this first step is a priority queue of seed areas, each one comprising a set of points. In the second step, seed areas are extracted from the queue according to their score. Whenever an area is popped, one of its points is selected randomly, and is treated as a new seed. The intuition is that seeds drawn from areas already having a relatively high score are more likely to lead to high-scoring regions. This seed is then expanded, using procedure ExpandBest which searches the space more thoroughly, until the maximum size M is reached or no more neighbors exist. At the end, the selected seed is removed from the set of points of that area, and the area is pushed back to the queue, with an updated priority score, which is the one found during the expansion process with ExpandBest. Hence, if the seed selected from that area has produced a high-scoring region, additional seeds will be drawn from the same area, examining it more exhaustively; otherwise, the priority of that area will become lower, leading the algorithm to draw seeds, and thus explore, an alternative one that may turn out to be more promising.

Table 2: POI datasets used in the experiments.

Dataset	*# points*	*avg. degree*
Athens	20,848	92
London	85,178	79
New York City	38,843	80

Adaptive Grid. The second algorithm, called AdaptiveGrid, also combines ExpandAll and ExpandBest in a two-phase process. In the first phase, a low-granularity grid is constructed on the space. Then, a small number of seeds is considered by first picking a grid cell uniformly at random, and then choosing a point within the cell also uniformly at random. The grid serves two purposes: (i) to locate seeds that are uniformly distributed in space and thus avoid oversampling/undersampling a dense/sparse area; and, more importantly, (ii) to identify good seed areas. At each selected seed, ExpandAll is executed, and the resulting score is attributed to the cell where the seed belongs. After the first phase is completed, the cells with the k highest scores are qualified as the seed areas to be examined in the second phase. In the second phase, a higher granularity grid is constructed within each qualified seed area. Then, a series of seed probes is performed until the time budget is exceeded. Each seed probe entails three steps, the selection of a seed area, the selection of a cell within the high-granularity grid of the seed area, and the selection of a point within that cell; all selections are done uniformly at random. A seed selected in this phase will be expanded more thoroughly using procedure ExpandBest.

5 EXPERIMENTAL EVALUATION

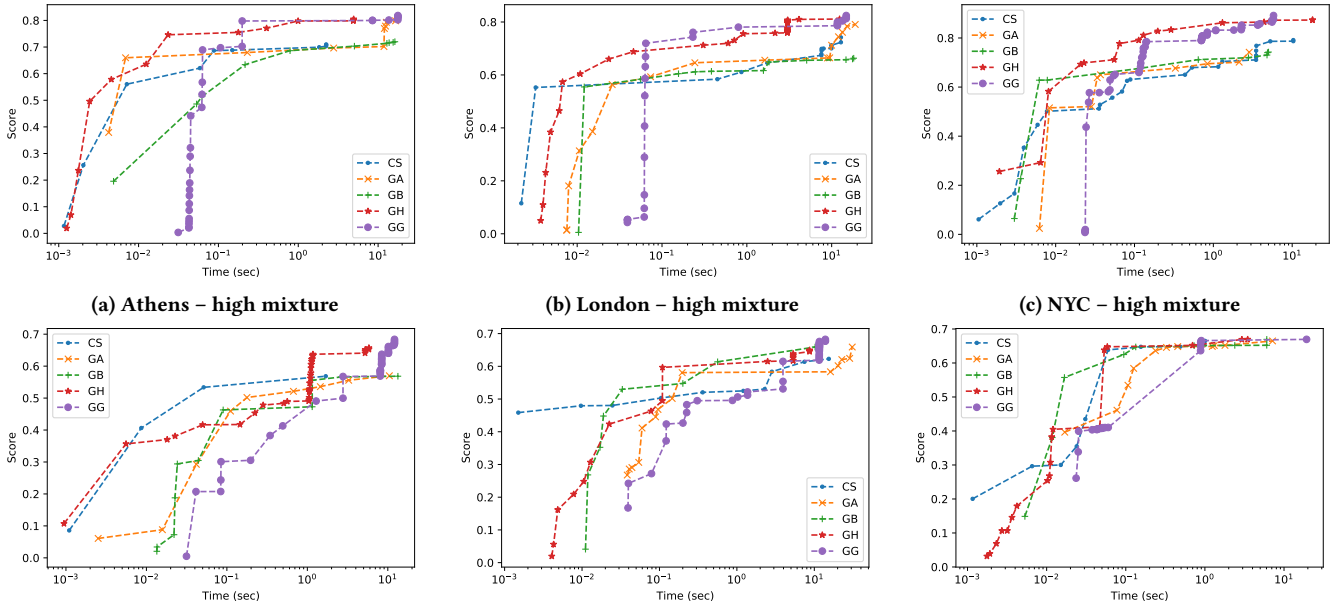
5.1 Experimental Setup

Datasets. We have conducted experiments using three real-world datasets containing POIs belonging to 15 different categories (e.g., transport, shop, food, education, tourist, etc.) extracted from OpenStreetMap. These datasets cover the metropolitan areas of Athens, London and New York, and are listed in Table 2, where the last column refers to the average node degree in the spatial connectivity graph G. The latter was constructed with $\epsilon = 0.003°$ to also accommodate areas outside the city center, where POIs are sparser.

Parameters. In the conducted experiments, we have tested the effect of several parameters involved in the performance of the

(a) CS – low mixture (b) GG – low mixture (c) CS – high mixture (d) GG – high mixture

Figure 2: Indicative best regions discovered in Athens (same map scale per mixture mode).



(a) Athens – high mixture (b) London – high mixture (c) NYC – high mixture

(d) Athens – low mixture (e) London – low mixture (f) NYC – low mixture

Figure 3: Score vs. time for each mixture type per dataset.

Table 3: Parameters used in the experiments.

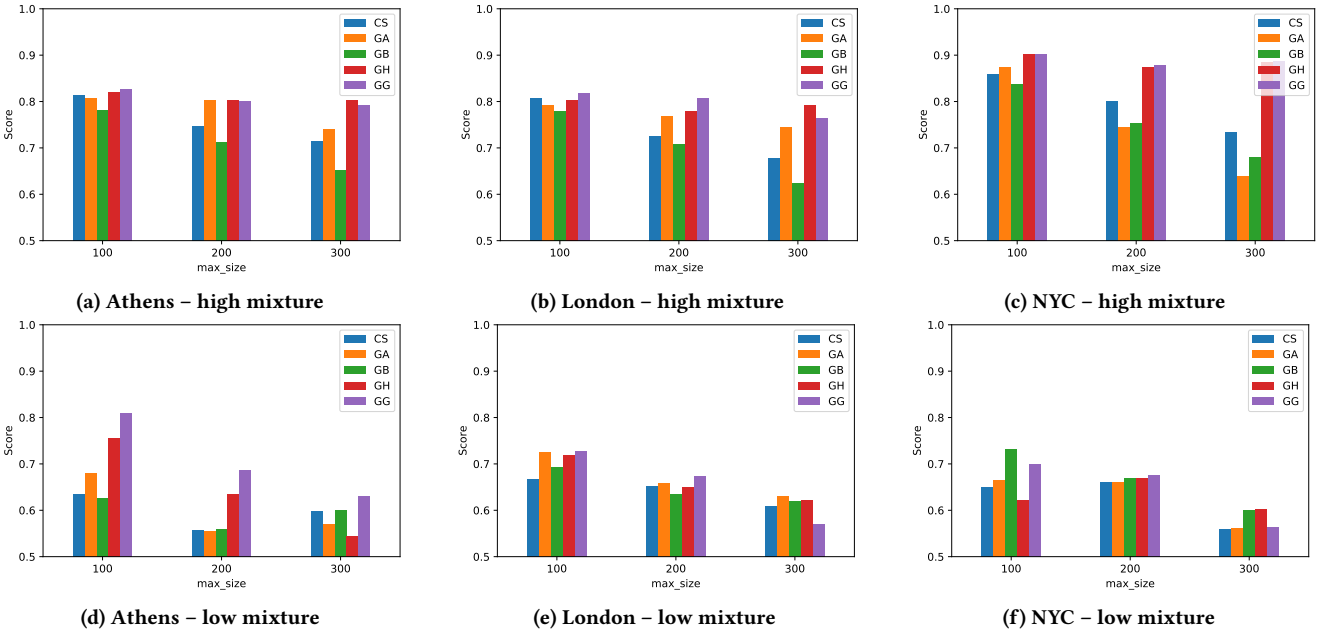
Parameter	*Values*
max region size M	100, **200**, 300
size weight γ	0, 0.5, **1**, 2
time budget T	**20**, 60
type of mixture	high, low

algorithms. These include: (i) the maximum region size M; (ii) the weight γ of the region size in Equation 1; (iii) the time budget T for the execution of each algorithm; and (iv) the type of mixture (high/low) we are interested in. Table 3 lists the range of parameter values tested, with default values shown in bold.

Methods. We compare the following methods: (i) CS – the algorithm CircularScan, described in Section 4.1; (ii) GA – the algorithm GraphExpand configured to use the ExpandAll strategy (see Section 4.2); (iii) GB – the algorithm GraphExpand configured to use the ExpandBest strategy (see Section 4.2); (iv) GH – the algorithm AdaptiveHybrid in Section 4.3; and (v) GG – the algorithm AdaptiveGrid also described in Section 4.3. For each dataset and mixture mode, we select the best parameterization of each method with a tuning process discussed in Appendix A.2.

All algorithms were implemented in Python, and the code is available on GitHub.[1] The experiments were executed on a server with AMD Ryzen Threadripper 3960X 24-Core processor and 256 GB RAM running Ubuntu 20.04.

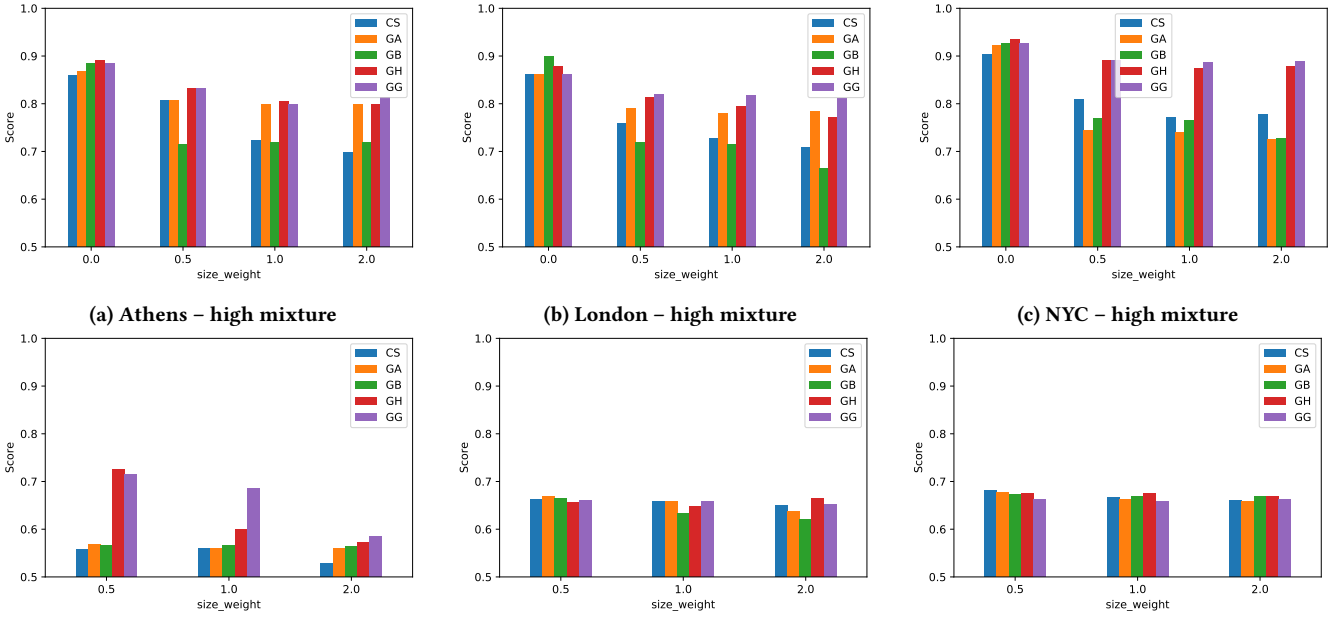
[1] [FIXME: TODO]



(a) Athens – high mixture (b) London – high mixture (c) NYC – high mixture

(d) Athens – low mixture (e) London – low mixture (f) NYC – low mixture

Figure 4: Score per dataset with varying max size M of best regions.



(a) Athens – high mixture (b) London – high mixture (c) NYC – high mixture

(d) Athens – low mixture (e) London – low mixture (f) NYC – low mixture

Figure 5: Score per dataset with varying size weight γ.

5.2 Qualitative Results

Before comparing the performance of the different algorithms, we present indicative results to qualitatively assess the detected regions. We present map plots of indicative best regions detected from algorithms in the various datasets under the default parameter and configuration settings. More specifically, we visually compare regions discovered by the baseline CS with those identified by GG.

We discuss the findings from Athens here, and defer discussion about the other datasets in Appendix A.1. As depicted in Figure 2a, the *low mixture* region returned by CS contains much less than $M = 200$ POIs scattered in a small area around one of the initial

seeds. Instead, the region found by GG (Figure 2b) has a non-convex shape and covers a much greater area (both maps are in the same scale). Similar observations can be drawn regarding *high-mixture* regions: the compact region detected by CS (Figure 2c) is small in size and extent, while the region issued by GG (Figure 2d) includes POIs along a highway and several districts nearby.

These differences are due to the fact that CS detects regions using a fixed shape (circle), so usually it cannot significantly enlarge the extent of a candidate region without compromising the quality of its mixture pattern. Instead, in GG, the region can adapt its shape to better capture the underlying mixture pattern, thus detecting larger areas while still satisfying cohesiveness and completeness.

5.3 Performance Evaluation

Comparison of anytime behavior. In this part of the experiments, we compare the different algorithms with respect to their ability to detect high-scoring regions under any given time budget. We set all parameters to their default values, and we monitor the score of the best region found by each algorithm at any given point in time up to a limit of 20 seconds. The results for each dataset and for each type of mixture are presented in Figure 3. The most important observations are the following. In all settings, the two-step adaptive methods, GH and GG, identify regions with the best scores. The non-adaptive methods, CS, GA, and GB, can relatively quickly identify good regions, but they often reach a plateau beyond which they cannot improve the quality of the region found; CS demonstrates this behavior most often. In contrast, GH is able to identify early on good regions and improve upon them. GG is slightly slower to identify a good region (due to an overhead of around 0.1 sec to build the grid), but is able to improve upon its solutions much faster. In conclusion, for a time budget in the order of hundredths of a second, GH consistently identifies better regions. If a few seconds can be afforded, GG is often the best option.

Impact of parameters. In the rest of the experiments, we investigate the effect of the parameters in the performance of the compared algorithms, namely the maximum region size M and the weight of the region size γ. The results for the different datasets and mixture types are presented in Figures 4 and 5. It is worth highlighting that in almost all settings, the adaptive seed prioritization of GH and GG leads to considerable improvements over the fixed-seed graph expansion methods. Moreover, in most cases, for the GraphExpand method, the ExpandAll strategy (GA) outperforms ExpandBest (GB). This seems counter-intuitive, since GB is a more thorough expansion meaning that given a seed it will most probably discover a higher-scoring region. However, the explanation is that GB is slower than GA in expanding a seed (in our experiments, GA is about four times faster); hence, given the same time budget, GA is able to consider more seeds than GB. This advantage of GA over GB is explicitly exploited by the adaptive methods, which seek to first quickly identify good seed areas, and then exploit them.

In terms of the effect of M, larger areas make the search harder for high mixture, resulting in a greater gap between GH/GG and CS/GA/GB. Indeed, there are more opportunities to create a high mixture area with more points considered. Conversely, size M has smaller effect on the relative merit of the various methods in low mixture. Similar observations hold for the effect of the size weight γ. When γ increases, larger regions are preferred, which makes the problem for high mixture harder, and necessitates adaptive methods. The effect of γ for the low mixture problem is moderate.

6 CONCLUSIONS

We have studied the problem of discovering spatial regions characterized by high or low mixture of enclosed point types. Unlike existing works that detect regions of fixed shape, like circle or rectangle, our approach can discover regions of arbitrary shape, thus better reflecting real-world spatial patterns. We introduce anytime algorithms that employ subgraph expansion to prioritize generation and evaluation of candidate regions under a given time budget. Our experimental evaluation on real-world POI datasets shows that better regions can be detected even within tight time constraints.

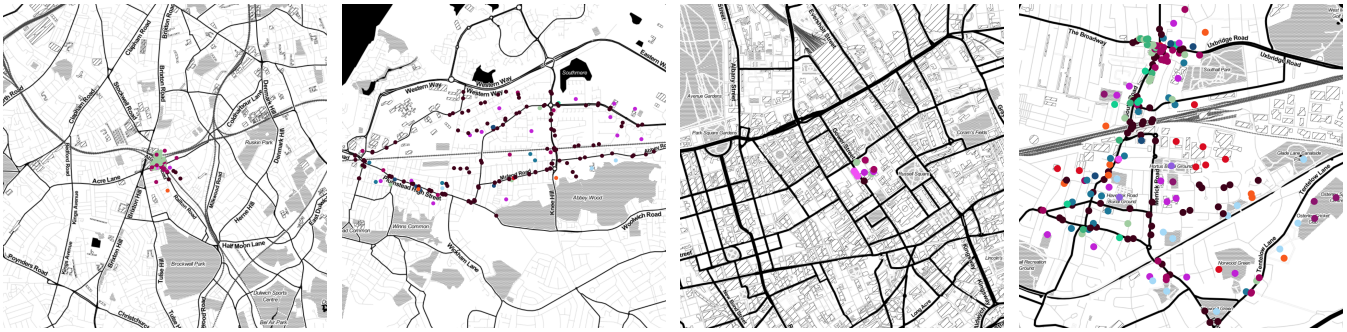
In the future, we plan to investigate methods for obtaining diversified top-k results, as well as measuring statistical significance. We will also experiment with additional types of data, e.g., types of crime incidents or topics from geolocated tweets and photos.

ACKNOWLEDGMENTS

This work was supported by the EU H2020 project SmartDataLake (825041).

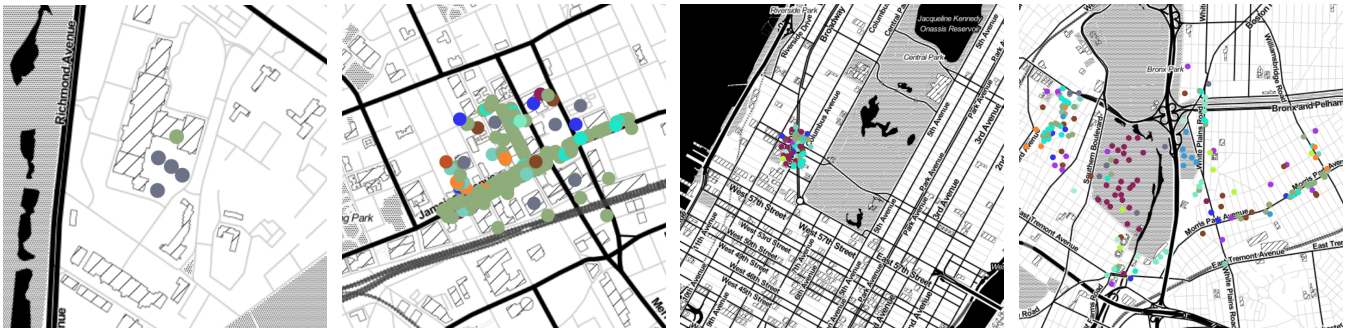
REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*. 487–499.
[2] Daichi Amagata and Takahiro Hara. 2017. A General Framework for MaxRS and MaxCRS Monitoring in Spatial Data Streams. *ACM TSAS* 3, 1 (2017), 1:1–1:34.
[3] Luc Anselin. 1995. Local indicators of spatial association—LISA. *Geographical analysis* 27, 2 (1995), 93–115.
[4] Spiros Athanasiou, Giorgos Giannopoulos, Damien Graux, Nikos Karagiannakis, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Kostas Patroumpas, Mohamed Ahmed Sherif, and Dimitrios Skoutas. 2019. Big POI data integration with Linked Data technologies. In *EDBT*. 477–488.
[5] Xin Cao, Gao Cong, Tao Guo, Christian S. Jensen, and Beng Chin Ooi. 2015. Efficient Processing of Spatial Group Keyword Queries. *ACM Trans. Database Syst.* 40, 2 (2015), 13:1–13:48.
[6] Xin Cao, Gao Cong, Christian S. Jensen, and Man Lung Yiu. 2014. Retrieving Regions of Interest for User Exploration. *Proc. VLDB Endow.* 7, 9 (2014), 733–744.
[7] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. 2012. A Scalable Algorithm for Maximizing Range Sum in Spatial Databases. *PVLDB* 5, 11 (2012), 1088–1099.
[8] Dong-Wan Choi, Jian Pei, and Xuemin Lin. 2016. Finding the minimum spatial keyword cover. In *ICDE*. 685–696.
[9] Ke Deng, Xin Li, Jiaheng Lu, and Xiaofang Zhou. 2015. Best Keyword Cover Search. *IEEE Trans. Knowl. Data Eng.* 27, 1 (2015), 61–73.
[10] Ahmed Eldawy and Mohamed F. Mokbel. 2016. The Era of Big Spatial Data: A Survey. *Found. Trends Databases* 6, 3-4 (2016), 163–273.
[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.
[12] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. 2016. Towards Best Region Search for Data Exploration. In *SIGMOD*. 1055–1070.
[13] Joseph Glaz and Markos V. Koutras. 2019. *Handbook of Scan Statistics*. Springer.
[14] Inkyung Jung, Martin Kulldorff, and Otukei John Richard. 2010. A spatial scan statistic for multinomial data. *Statistics in medicine* 29, 18 (2010), 1910–1918.
[15] Hamid Shahrivari, Matthaios Olma, Odysseas Papapetrou, Dimitrios Skoutas, and Anastasia Ailamaki. 2020. A Parallel and Distributed Approach for Diversified Top-k Best Region Search. In *EDBT*. 265–276.
[16] Dimitrios Skoutas, Dimitris Sacharidis, and Kostas Patroumpas. 2018. Efficient progressive and diversified top-k best region search. In *SIGSPATIAL*. 299–308.
[17] Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. 2013. Approximate MaxRS in Spatial Databases. *Proc. VLDB Endow.* 6, 13 (2013), 1546–1557.
[18] Yiqun Xie, Han Bao, Yan Li, and Shashi Shekhar. 2020. Discovering Spatial Mixture Patterns of Interest. In *SIGSPATIAL*. 608–617.
[19] Yiqun Xie and Shashi Shekhar. 2019. Significant DBSCAN towards Statistically Robust Clustering. In *SSTD*. ACM, 31–40.
[20] Yiqun Xie, Shashi Shekhar, and Yan Li. 2021. Statistically-Robust Clustering Techniques for Mapping Spatial Hotspots: A Survey. *CoRR* abs/2103.12019 (2021).



(a) CS – low mixture (b) GG – low mixture (c) CS – high mixture (d) GG – high mixture

Figure 6: Indicative best regions discovered in London (same map scale per mixture mode).



(a) CS – low mixture (b) GG – low mixture (c) CS – high mixture (d) GG – high mixture

Figure 7: Indicative best regions discovered in New York City (same map scale per mixture mode).

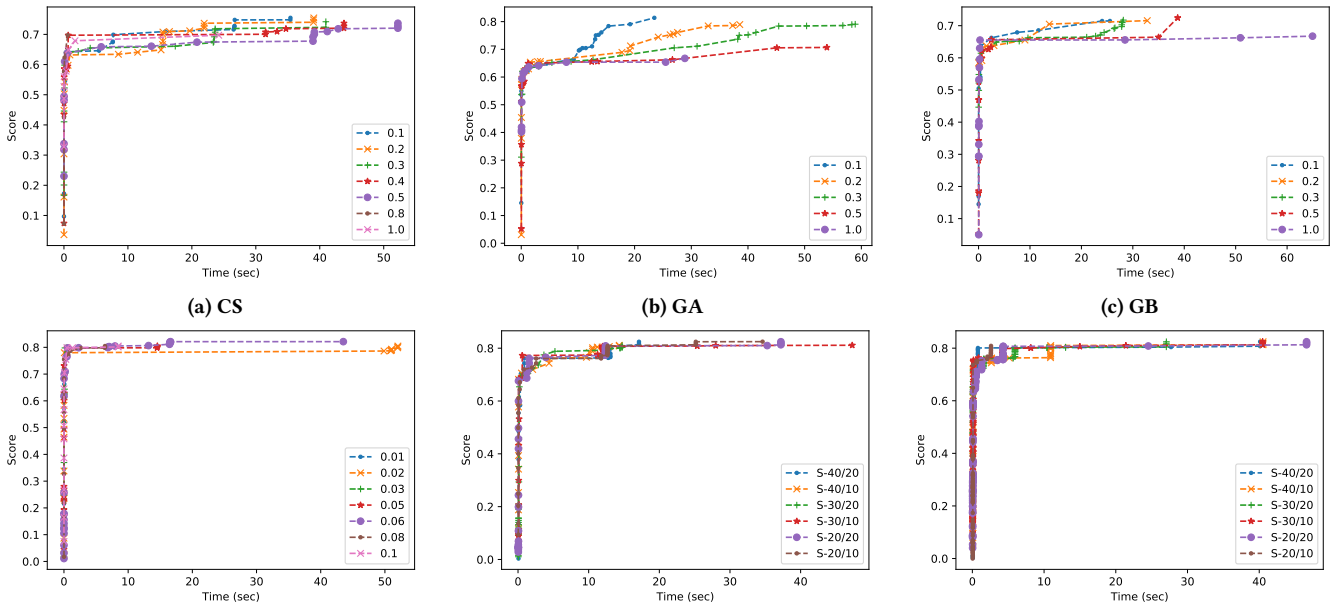
A APPENDIX

A.1 More Qualitative Results

In London, method CS identifies a group of shops in a mall as the best low mixture region (Figure 6a), surrounded by a few POIs included from other categories due to proximity. This area is very small compared with the one identified by GG (Figure 6b), which mainly comprises bus stops and metro stations along major roads. The best high mixture region identified by CS contains few POIs of different categories; instead, GG offers an extended region of size almost M as intended, and also achieves a higher score. Note that the area has a peculiar shape with varying density of POIs therein. Figure 7 illustrates the corresponding regions discovered in New York City. Clearly, GG successfully identifies a low mixture region densely dominated by many shops along a highway with only a few nearby POIs of assorted categories (Figure 7b). In contrast, CS gives a poor result with the suggested region containing a handful of similar POIs only (Figure 7a). Indeed, this method seems tightly bound to the initial seeds and regions grow circularly around them, so its chances to find a larger region are limited. For high mixture, CS indeed manages to find a good result (Figure 7c), but the randomness in the choice of seeds may yield poorer results in another execution. Again, the result from GG in Figure 7c stretches over a much greater area in an arbitrary fashion and includes more POIs.

A.2 Algorithm Tuning

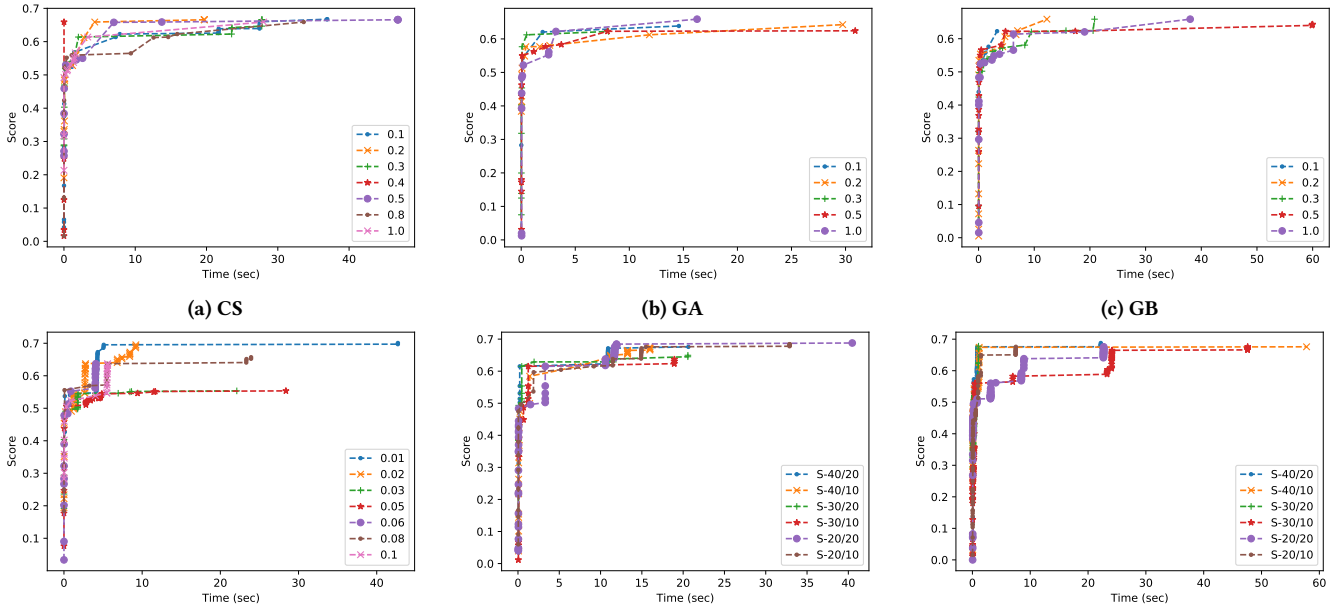
All algorithms presented in Section 4 involve some configuration parameters that control their anytime performance, i.e., how good and how fast regions are discovered. So, before comparing the various methods to each other, some tuning is required. For each method, and for each dataset and mixture mode (high/low), we execute the method with several different configurations, and allocate a time budget of 60 seconds. Upon termination, we monitor the score of the best detected region at each point in time. By inspecting the results for each algorithm, we find the configuration that performs the best around the 20 seconds mark — the default time budget. As an indication, Figures 8 and 9 present the score per time in Athens. Each plot investigates different configurations of one method. For example, Figure 8a considers the CS method, and plots the score over time curve for different seed ratios, starting from 0.1 up to 1. From this plot, we deduce that around the 20 seconds mark, the highest scoring region is identified by CS when configured for a seed ratio of 0.2. Similarly, Figures 8b–8d explore different seed ratios for GA, GB, and GH. The last two figures consider different configurations of the GG method. The first is when the low-granularity grid is explored via ExpandAll, and the second when explored via ExpandBest. The two numbers, like 40/20, mean a 40×40 grid where the top-20 cells are further explored.



(a) CS (b) GA (c) GB

(d) GH (e) GG-a (f) GG-b

Figure 8: Fine-tuning per method for high mixture mode in Athens.



(a) CS (b) GA (c) GB

(d) GH (e) GG-a (f) GG-b

Figure 9: Fine-tuning per method for low mixture mode in Athens.