A COMPUTATIONAL FRAMEWORK FOR FINDING INTERESTINGNESS HOTSPOTS IN SPATIAL DATASETS

A Dissertation Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Fatih Akdag

December 2016

A COMPUTATIONAL FRAMEWORK FOR FINDING INTERESTINGNESS HOTSPOTS IN SPATIAL DATASETS

Fatih Akdag

APPROVED:

Dr. Christoph F. Eick, Chairman

Dr. Edgar Gabriel

Dr. Guoning Chen

Dr. Thamar Solorio

Dr. Yunsoo Choi

Dean, College of Natural Sciences and Mathematics

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Christoph F. Eick, who has supported me throughout my dissertation with his guidance and knowledge. I would also like to thank my dear parents and my dear friends who supported me, and incented me to strive towards my goal.

My deepest appreciations to my wife, Tuba, for her help and patience that made it possible for me to complete this dissertation while working full time. A special thank is devoted to my lovely son Salih Efe for being such a powerful source of inspiration and energy.

A COMPUTATIONAL FRAMEWORK FOR FINDING INTERESTINGNESS HOTSPOTS IN SPATIAL DATASETS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Fatih Akdag

December 2016

ABSTRACT

The significant growth of spatial data increased the need for automated discovery of spatial knowledge. An important task when analyzing spatial data is hotspot discovery. In this dissertation, we propose a novel methodology for discovering interestingness hotspots in spatial datasets. We define interestingness hotspots as contiguous regions in space which are interesting based on a domain expert's notion of interestingness captured by an interestingness function. We propose computational methods for finding interestingness hotspots in point-based and polygonal spatial datasets, and gridded spatial-temporal datasets. The proposed framework identifies hotspots maximizing an externally given interestingness function defined on any number of spatial or non-spatial attributes using a five-step methodology, which consists of:

(1) identifying neighboring objects in the dataset,

(2) generating hotspot seeds,

(3) growing hotspots from identified hotspot seeds,

(4) post-processing to remove highly overlapping neighboring redundant hotspots, and

(5) finding the scope of hotspots.

In particular, we introduce novel hotspot growing algorithms that grow hotspots from hotspot seeds. A novel growing algorithm for point-based datasets is introduced that operates on Gabriel Graphs, capturing the neighboring relationships of objects in a spatial dataset. Moreover, we present a novel graph-based post-processing

algorithm, which removes highly overlapping hotspots and employs a graph simplification step that significantly improves the runtime of finding maximum weight independent set in the overlap graph of hotspots. The proposed post-processing algorithm is quite generic and can be used with any methods to cope with overlapping hotspots or clusters. Additionally, the employed graph simplification step can be adapted as a preprocessing step by algorithms that find maximum weight clique and maximum weight independent sets in graphs. Furthermore, we propose a computational framework for finding the scope of two-dimensional point-based hotspots.

We evaluate our framework in case studies using a gridded air-pollution dataset, and point-based crime and taxicab datasets in which we find hotspots based on different interestingness functions and we give a comparison of our framework with a state-of-the-art hotspot discovery technique. Experiments show that our methodology succeeds in accurately discovering interestingness hotspots and does well in comparison to traditional hotspot detection methods.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The scale of spatial data has grown rapidly with the widespread use of location-aware devices and data collection techniques. Extracting knowledge from large amounts of spatial data is becoming more important than before. Spatial data mining is the process of discovering interesting and previously unknown, but potentially useful patterns from spatial databases; however, the complexity of spatial data and implicit spatial relationships limits the usefulness of conventional data mining techniques for extracting spatial patterns [Shekhar et al. 2011].

An important task when analyzing spatial data is hotspot discovery. Shekhar et al. [2011] defines hotspot discovery in spatial data mining as "a process of identifying spatial regions where more events are likely to happen, or more objects are likely to appear, in comparison to other areas". Applications of hotspot analysis include crime analysis, epidemiology, voting pattern analysis, economic geography, retail analysis, traffic incident analysis, and demographics [Eftelioglu et al. 2016]. Eck et al. [2005] detects hotspots with high crime rates to help police identify high-crime areas, types of crime being committed, and the best way to respond. Kulldorff et al. [2005] uses hotspot analysis for early detection and monitoring of disease outbreak locations. Hale et al. [2015] uses hotspot analysis to determine the locations where nutrient yields and nutrient retentions occur. Barrell et al. [2013] identifies hotspots of high and low sea grass cover in the seabed. Yabe et al. [2016] detect evacuation hotspots after a large-scale disaster —the Kuromoto Earthquake—using location data from smartphones.

Similar to Shekhar's definition given earlier, Zhang and Eick [2016] define hotspots as dense regions in a spatial dataset whose density is above a user-defined threshold. On the other hand, Agarwal et al. [2006] defines hotspots as anomalous regions in spatial data, and considers hotspot discovery problem as finding maximum discrepancy regions.

In this research, we propose an alternative hotspot discovery approach, which aims to find hotspots in spatial datasets maximizing an externally given interestingness function based on a domain expert's notion of interestingness. In particular, we present a computational framework which grows hotspots from seed hotspots using an interestingness function. Interestingness functions are defined on the spatial and non-spatial attributes of the data and are used to measure the "hotness" or "news-worthiness" of a spatial region. This research extends the scope of hotspot discovery as it enables detecting hotspots based on an external interestingness function rather than using the number of events in a region as a measure of hotness, and enables finding hotspots that cannot be discovered by classical hotspot discovery techniques. For example, using our framework, we were able to identify with high correlation hotspots between PM2.5 and ozone levels in a four-dimensional gridded air pollution dataset [Akdag et al. 2014]. We define interestingness hotspots as contiguous regions in space, in which the interestingness value, computed by the plugin interestingness function, is larger than a predefined interestingness threshold. In general, our approach is capable of discovering regions with interesting patterns involving one or more spatial or non-spatial attributes.

There is significant work in using spatial scan statistics [Kulldorff 1997] for finding hotspots in spatial datasets. Spatial scan statistics based hotspot detection techniques

search for regions in the dataset with statistically high number of occurrences of an event compared to the rest of the dataset. When using these algorithms, each object's interestingness value—which is a dedicated attribute of an object—should be predefined and independent of properties of other objects in the hotspot region. Thus, they are unable to find interestingness hotspots whose value depends on properties of other objects in the hotspot region; for example, they cannot find hotspots where any given two attributes are highly correlated or where an attribute has very low variance, as correlation and variance measures are defined on a set of objects—rather than being an attribute of each object in the dataset. Furthermore, most spatial scan methods are restricted to finding hotspots with a certain shape, whereas our framework provides a graph-based hotspot growing approach that is quite general and extendible, and can identify hotspots of any shape.

Moreover, spatial clustering algorithms have been used for hotspot detection. However, most clustering algorithms compute clusters solely relying on distance information. Moreover, clustering algorithms search for all hotspots in parallel, being forced to make compromises—for example, switching a sub region from one to another cluster; such a switch might increase the reward of one cluster but decrease the reward of the other cluster. On the other hand, hotspot discovery algorithms grow hotspots serially from hotspot seeds without having to make any compromises; this enables, we claim, hotspot discovery algorithms to find "better" hotspots compared to clustering approaches. However when using hotspot discovery approaches, many of the hotspots obtained may overlap. Thus, the hotspot discovery approaches have to deal with overlapping hotspots, which requires a post-processing step that is not necessary for the clustering approaches that produce disjoint clusters.

This research focuses on finding interestingness hotspots in point-based and polygonal spatial datasets and gridded spatial-temporal datasets. Naturally, most spatial datasets are point-based datasets where each object corresponds to a location. Polygons are also used extensively for representing two-dimensional spatial objects such as cities, regions, states, countries, zip codes, etc. Gridded spatial datasets are quite common in scientific computing as many disciplines such as optometry, Earth and atmospheric sciences, medicine, and ecology produce large amounts of data relying on spatial grid-structures that identify locations where measurements are taken.

In addition, we propose a methodology for finding the scope for point-based hotspots using polygon models. We propose two methods for creating polygons models for point-based hotspots, and clusters. The first method creates polygons based on Voronoi diagram and convex hull of the point set. The second method is a more generic approach that creates tighter polygons based on Delaunay triangulation of the dataset, and uses a novel fitness function to select the parameter for generating a polygon model and does not require access to the whole dataset.

In summary, we propose a framework that identifies hotspots maximizing an externally given interestingness function using a five-step approach, which consists of:

(1) Identifying neighboring objects in the dataset,

(2) Identifying hotspot seeds,

(3) Growing hotspots from identified hotspot seeds,

(4) Post-processing to remove highly overlapping redundant hotspots,

(5) Finding the scope of each hotspot.

The main contributions of this dissertation include:

—We present novel hotspot growing algorithms, which grow interestingness hotspots from seed hotspots

—We introduce novel interestingness functions that can be of interest to domain experts.

—We present a novel graph-based post-processing algorithm, which removes highly overlapping redundant hotspots by finding maximum weight cliques in overlap graphs. This algorithm is generic and can be used with any algorithm that creates overlapping hotspots or clusters.

—We introduce a graph simplification algorithm that significantly improves the runtime of finding maximum weight independent set and maximum weight cliques in overlap graphs.

—Moreover, we present a novel algorithm to determine the scope of interestingness hotspots that that uses polygons as hotspot models which are computed from Voronoi diagram and the convex hull of the objects that belong to the hotspot.

—We present an alternative methodology for creating polygon models for spatial hotspots and clusters. As part of this approach, we propose a novel "polygon-emptiness measure" that is used to assess the amount of empty spaces in polygons generated for a set of points.

—We provide case studies in which we find:

  —hotspots in an air-pollution dataset with high correlation of air pollutants and ozone levels,

—hotspots in a crime dataset where majority of crimes belong to the same crime type,

—hotspots in a taxicab dataset where more money is made by taxi drivers per minute that pick up passengers in the hotspot region, and

—polygon models for a set of clusters in an artificial dataset

The remainder of the dissertation is organized as follows: Chapters 2 and 3 introduce the background and the related work, respectively. Chapter 4 describes the proposed hotspot discovery framework. Chapter 5 gives a detailed explanation of the methodology we use. In Chapter 6, we evaluate our framework in multiple case studies and compare our framework with the state of art hotspot discovery approaches. Chapter 7 gives a conclusion of the dissertation.

CHAPTER 2

BACKGROUND

In this chapter, we firstly introduce the basic concepts of graph theory that are relevant in this dissertation. Next, we give a description of some commonly used data structures and computational geometry concepts that are frequently used in our research.

2.1 Graph Theory

In this section, we introduce the basic concepts of graph theory that are relevant in this dissertation.

Definition 2.1 (Graph). A **graph** G is a pair of sets (V,E), where V is a set of vertices, and E is a set of edges between the vertices, where $E \subseteq \{(u,v) \mid u, v \in V\}$. The vertices may be part of the graph structure, or may be external entities represented by integer indices or references. Two vertices are *adjacent* if they are connected to each other through an edge. A weighted graph is a graph whose vertices or edges have associated weights. More specifically, a vertex-weighted graph has weights on its vertices and an edge-weighted graph has weights on its edges. When the edges in a graph have a direction, the graph is called a directed graph or digraph, and the edges are called directed edges.

Graphs can be represented using different data structures:

(1) Adjacency lists: Vertices are stored as objects, and every vertex keeps a list of adjacent vertices.

(2) Adjacency matrix: A two-dimensional matrix is used to store vertices and edges, in which the rows represent source vertices and columns represent destination vertices.

(3) Incidence matrix: A two-dimensional boolean matrix in which the rows represent the vertices and columns represent the edges. The entries indicate whether the vertex at a row is incident to the edge at a column.

Definition 2.2 (Complete graph). A **complete graph** is an undirected graph in which every pair of distinct vertices is connected by an edge (Figure 1).

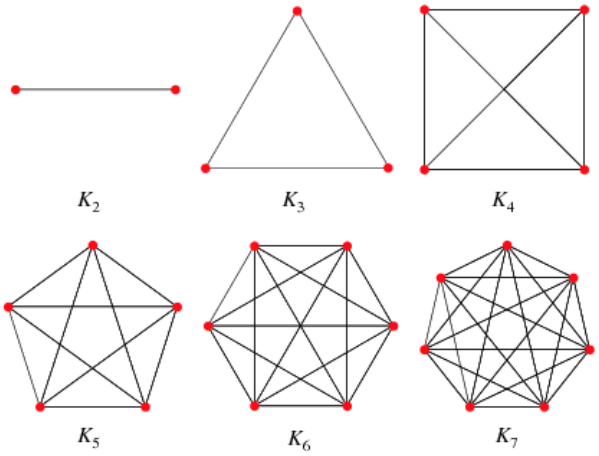


Figure 1. Complete graphs of various sizes

Definition 2.3 (Clique). A **clique** is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. Each vertex and edge in a graph is also considered cliques of size 1 and 2 respectively. Finding if there is a clique of a given size in a graph is called the clique problem, it is one of the well-known and hardest NP-complete problems.

Definition 2.4 (Maximum weight clique). The clique with the maximum weight in a vertex-weighted graph is called a "**maximum weight clique**". Finding the "maximum weight clique" is an NP-hard problem [Gary and Johnson 1979].

Definition 2.5 (Complement graph). The **complement** of a graph G is a graph G' on the same set of vertices such that two distinct vertices of G' are adjacent if and only if they are not adjacent in G.

Definition 2.6 (Independent set). An **independent set** is a set of vertices in a graph, in which none of the vertices are adjacent (Figure 2). Cliques and independent sets are complimentary. The complement of a clique is an independent set and the complement of an independent set is a clique.

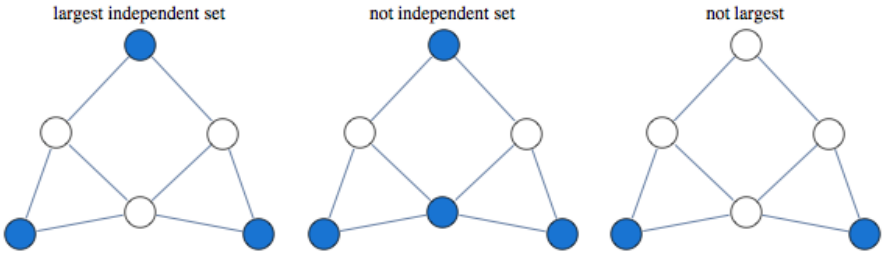


Figure 2. Independent set in a graph

Definition 2.7 (Maximum weight independent set). The independent set with the maximum weight in a vertex-weighted graph is called a "**maximum weight independent set**", and finding the "maximum weight independent set" is a dual of "maximum weight clique problem" and similarly it is an NP-hard [Gary and Johnson 1979] problem.

Definition 2.8 (Connected component). A **connected component** of an undirected graph G is a sub graph C in which all pair of vertices are connected to each other by paths, and not connected to other vertices in the graph.

2.2 Data Structures

In this section, we give a brief description of the basic data structures that are used extensively in this dissertation.

2.2.1 Max-heap

A max-heap is a tree-based data structure that satisfies the heap ordering property: The value (priority) of each node is always smaller than or equal to the value of its parent (Figure 3). The node with the highest priority is always at the root. The max-heap data structure is a maximally efficient implementation of a priority queue.

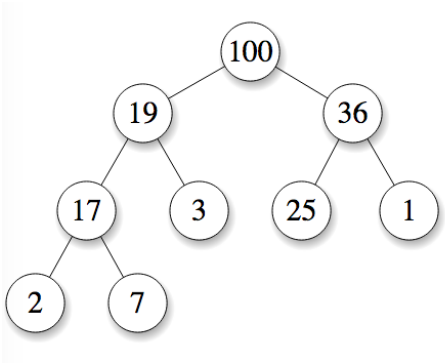


Figure 3. Representation of a max-heap

Heap data structure has optimal O(1) time complexity for find-max and insert-node operations and O(log$n$) complexity for delete-max operation. However, it is inefficient for find-node operation on a random node, which has O(n) time complexity.

2.2.2 Hash Set

A hash set is a high-performance implementation of a collection holding a set of objects. A set is a collection that contains no duplicate elements, and whose elements are in no particular order (not sorted). A hash table is used to store objects in the set

by using a hash function that maps objects to indices in the hash table (Figure 4). Hash set data structure has an optimal O(1) time complexity for add, remove, and contains operations, whereas it is not efficient in terms of storage, sorting and iterating over objects. Hash set provides mathematical set operations such as set union, intersect, subtract, and overlaps.

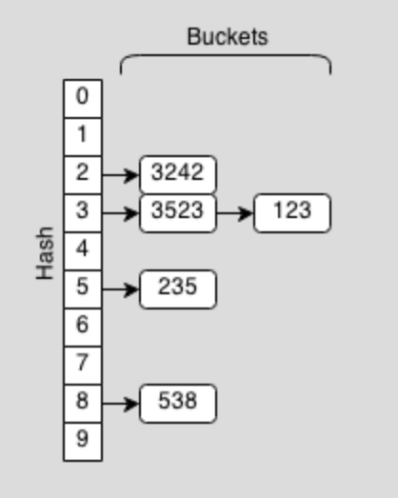


Figure 4. Representation of a hash set

2.3 Computational Geometry

Many concepts in computation geometry plays an important role in spatial data mining. In this section, we describe the computational geometry concepts used in this dissertation.

2.3.1 Delaunay Triangulation

Delaunay triangulation (DT) for a set of points P in a plane is a triangulation such that no point in P is inside the circumcircle of any triangle in DT(P). Two points p_i and p_j are connected by an edge in the Delaunay triangulation, if and only if there is an empty

circle passing through p_i and p_j (Figure 5). The exterior face of the Delaunay triangulation is the **convex hull** of the point set. DT can be calculated in O($n$log$n$) time [Cignoni et al. 1998] where n is the number of points in the point set.

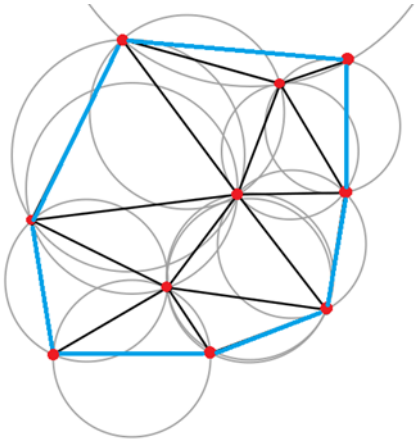


Figure 5. Delaunay triangulation and convex hull of a set of points.
Exterior face of the DT is the convex hull (shown in blue)

2.3.2 Convex Hull

Definition 2.9 (Convex set). A convex set is a region in Euclidian space such that, all points on all straight line segments that join the pair of points in the region is also within the region.

A convex hull for a set of points P in the Euclidian space is the smallest convex set that contains all points in P (blue lines in Figure 5). For two and three dimensional point sets, convex hull can be calculated in O($n$log$h$) time where n is the number of points in the point set, and h is the number of points on the convex hull. For higher dimensions, computation takes O($n$log$n$ + $n^{d/2}$), where d is the number of dimensions [Chazelle 1993].

2.3.3 Voronoi Diagram

Voronoi diagram of a set of points (sites) is a partitioning of the plane into convex polygons such that each polygon corresponds to exactly one point (site) on the plane and every point in the interior of a polygon is closer to the corresponding site than to any other site. Voronoi diagram is also known as Voronoi Tessellation and Dirichlet tessellation. The Voronoi diagram of a set of points is dual to the Delaunay triangulation for the point set; for every Delaunay triangulation there exists a corresponding Voronoi tessellation and vice versa (Figure 6). Voronoi diagram can be calculated in O($n$log$n$) time where n is the number of points.

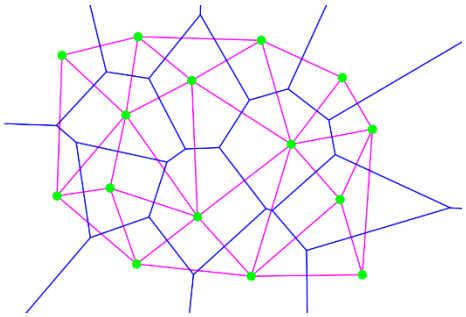
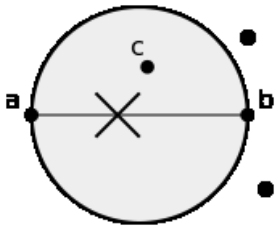


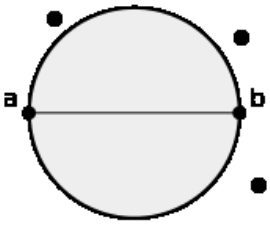
Figure 6. Voronoi diagram (blue) and Delaunay triangulation (pink) of a point set

2.3.4 Gabriel Graph

Gabriel graph [Gabriel and Sokal 1969] is a type of proximity graph, which is used to express the proximity of points. Two distinct points a and b in a point set are adjacent in the Gabriel graph if the closed disc d, of which the line segment ab is a diameter, contains no other points (Figure 7).



point c is inside the circle whose diameter is ab, so a and b are not Gabriel neighbors



no points inside the circle whose diameter is ab, so a and b are Gabriel neighbors

Figure 7. Diameter circle and its relation to Gabriel neighborhood

Unlike Delaunay graph, Gabriel graphs generalize to higher dimensions, with the empty disks replaced by empty closed balls. For a two-dimensional dataset with n points, the Gabriel graph can be computed from the Delaunay graph in O(n) time in a total of O($n$log$n$) complexity. For higher dimensional data, Gabriel graph can be computed in O(n^3) time by brute force. In a Gabriel graph, number of edges cannot be more than three times the number of vertices [Matula et al. 1980].

CHAPTER 3

RELATED WORK

Spatial scan statistics (SatScan) introduced by Kulldorff [1997] is the most popular hotspot discovery tool. It searches circular regions occurring within a certain time interval and can obtain circular hotspots by growing circles from a point of origin by increasing the radius of the circle. SatScan finds regions where there is an unusually high number of occurrence of an event compared to the rest of the dataset. In order to achieve this, it employs statistical significance tests to evaluate the statistical significance of the discovered hotspots. However, our goal is different, as we try to search for regions maximizing the plugin interestingness function rather than finding regions with unusually high number of occurrences of an event; therefore, we do not need to employ statistical significance tests. We require that the domain expert knows which patterns are interesting and incorporates that information in the interestingness function definition either as a parameter or as part of an expression. Thus, our approach is not an automatic hotspot discovery approach like SatScan. On the other hand, spatial scan statistics cannot be used to detect hotspots where the interestingness (e.g., hotness) of the region is defined based on a set of objects in the spatial region (e.g., correlation, variance, average, etc.) rather than being defined per object (e.g., a single attribute representing the interestingness value of object.)

SatScan was initially designed to find circular regions; however, spatial scan statistics were later extended for detecting ecliptic hotspots [Kulldorff et al. 2006], square pyramid shaped space-time hotspots [Iyengar, 2004], rectangular hotspots

[Neill et al. 2006], flexibly-shaped hotspots [Tango et al. 2005], irregular-shaped hotspots [Patil et al. 2004, Duczmal et al. 2004], ring-shaped hotspots [Eftelioglu et al. 2014], network hotspots with holes, linear hotspots [Eftelioglu et al. 2016], and significant routes [Oliver et al. 2014]. Moreover, various spatial correlation measures has been used with spatial statistics such as "Ripley's K Function, Moran's I, Local Moran Index, Getis Ord, Geary's C, etc." [Shekar et al. 2011]. Neill [2009] gives an empirical comparison of several of spatial scan statistics for outbreak detection, and [Eftelioglu et al. 2016] compares the capabilities of various spatial scan statistics for statistically significant crime hotspot detection. Our framework allows modeling a point, line, polygon, or grid cell, etc. as the base spatial object type, and allows a plugin neighborhood relation function to be used; therefore, any type of interestingness hotspot can be discovered using our approach.

Spatial scan statistics can be used to detect interestingness hotspots when the interestingness measure can be defined on each object independently of other objects in the dataset (e.g., as an input attribute). However, since the goal is to find statistically significant hotspots that have quite different statistical measures compared to the rest of the dataset, it is sensitive to extreme outliers. In other words, small hotspots will be detected around outliers, as outliers will cause extreme statistics in small regions.

There are also spatial clustering algorithms which can be used for computing spatial hotspots. Shekar et al. [2011] describe spatial clustering "a process of grouping a set of spatial objects into clusters so that objects within a cluster have high similarity in comparison to one another, but are dissimilar to objects in other clusters" and classifies common spatial clustering approaches as hierarchical, partitional and

density-based. The density-based clustering algorithm DBSCAN [Ester et al. 1996] has been used and extended by many for performing spatial clustering. Another popular clustering algorithm SNN [Ertoz et al. 2013] (Shared Nearest Neighbor) uses the number of shared neighbors in k-nearest neighbor lists to assess the similarity of spatial objects which enables the algorithm to identify clusters of varying densities. However, most clustering algorithms construct clusters solely based on distance information. They identify dense areas as hotspots and provide computational mechanisms to classify objects in non-dense areas as outliers. Most clustering algorithms cannot be used to detect interestingness hotspots based on a given interestingness function.

A new group of clustering algorithms has been introduced in the literature that find contiguous clusters by maximizing plug-in interestingness functions similar to the approach used in this dissertation. These algorithms are capable of considering non-spatial attributes in objective functions that drive the clustering process. Clusters are computed maximizing the sum of the rewards for each cluster based on a cluster interestingness function. CLEVER [Cao et al. 2013] is a k-medoids-style clustering algorithm which exchanges cluster representatives as long as the overall reward grows, whereas MOSAIC [Choo et al. 2007] and STAXAC [Amalaman and Eick 2015] are agglomerative clustering algorithms which start with a large number of small clusters, and then merge neighboring clusters as long as merging increases the overall interestingness.

Clustering algorithms search for all hotspots in parallel, being forced to make compromises, as switching from one sub region to another cluster might increase the reward of one cluster but decrease the reward of the other cluster. We grow hotspots

from seed regions independently, thus each hotspot can grow as much as it can without any compromises.

CHAPTER 4

INTERESTINGNESS HOTSPOT DISCOVERY FRAMEWORK

In this section, we give a description of the framework for discovering hotspots in spatial datasets using interestingness functions. Moreover, we introduce interestingness functions and neighborhood definitions that are implemented in our methodology.

4.1 A Framework for Spatial Interestingness Scoping

Interestingness hotspots are contiguous areas in space for which an interestingness function *i* assigns a *reward* w>0, indicating "news-worthy" regions in a spatial dataset. The scope of an interestingness hotspot is a contiguous spatial region whose interestingness is above a certain threshold θ.

More formally, we assume a spatial dataset O is given in which objects $o \in O$ are characterized by: a set of spatial attributes S, and a set of non-spatial attributes M. Moreover, we assume a spatial neighboring relationship N is given where $N \subseteq O \times O$; that describes which objects belonging to O are neighbors. N is usually computed using spatial attributes S of objects in O. Finally, we assume that we have an interestingness measure:

$$i: 2^O \rightarrow \{0\} \cup \Re^+$$

that assesses the interestingness of subsets of the objects in O by assigning rewards to a particular set of objects H. Moreover, we assume an interestingness threshold θ is given that defines which patterns are interesting.

The goal of this research is to develop frameworks and algorithms that find interestingness hotspots H⊆O; H is an interestingness hotspot with respect to i if the following 2 conditions are met:

(1) i(H) ≥θ

(2) H is contiguous with respect to N; that is, for each pair of objects (o,v) with o,v∈H, there has to be a path from o to v that traverses neighboring objects (w.r.t. N) belonging to H. In summary, interesting hotspots H are contiguous regions in space that are interesting (i(H) ≥ θ).

Moreover, our framework is also capable of finding spatial-temporal hotspots in gridded spatial-temporal datasets. Temporal attributes of grid cells are used for identifying neighboring grid cells in time dimension.

4.2 Example Interestingness Functions

A very simple interestingness measure is the one that directly uses the value of a single performance attribute p, which is defined as follows:

$$i_p(\mathrm{H}) = \frac{\sum_{h \in H} h.p}{|H|} \qquad (1)$$

where H⊆O is an interestingness hotspot, |H| denotes cardinality of H and h.p denotes the value for attribute p for object h in H. Many existing hotspot discovery techniques can be used to find hotspots based on $i_{p(}$H).

Another interestingness function considers the correlation of two performance attributes p_1 and p_2; the corresponding interestingness function $i_{corr}(p_1,p_2)$ is defined as follows:

$$i_{corr(p1,p2)}(H) = \begin{Bmatrix} 0, & if\ |correl(H,p_1,p_2)| < \theta \\ |correl(H,p_1,p_2)| - \theta, & otherwise \end{Bmatrix} \quad (2)$$

where $0 < \theta < 1$ is the interestingness threshold, and correl(H,p_1,p_2) is the correlation of attributes p_1 and p_2 with respect to the objects belonging to hotspot H. This interestingness function is used to find correlation hotspots .

The Variance interestingness function $i_{var}(p)$ operates on the variance of a non-spatial attribute p and finds high variance hotspots with respect to attribute p; it is defined as follows:

$$i_{var(p)}(H) = \begin{Bmatrix} \theta - variance(H,\mathrm{p}), & if\ variance(H,p) < \theta \\ 0, & otherwise \end{Bmatrix} \quad (3)$$

where $\theta > 0$ is the variance threshold, and variance(H,p) is the variance of an attribute p with respect to the objects that form hotspot H. This interestingness function is used to find regions in a dataset where an attribute p does not change significantly. The obtained hotspots can be used to generate maps for the attribute and for generating prediction models for the attribute—similar to regression trees.

Finally, we propose a *purity* interestingness function, which measures uniformity by the degree of dominance of instances belonging to a single category. The purity interestingness function is used for analyzing interestingness with respect to a categorical non-spatial attribute. Purity interestingness $i_{pur}(H)$ of a hotspot H is computed using the following formula:

$$i_{pur}(H) = \begin{cases} 0, & \max\left(prop_t(H)\right) < \theta \\ \left(\max\left(prop_t(H)\right) - \theta\right), & otherwise \end{cases} \quad (4)$$

where *c(H)* is the set of classes in the hotspot H, $t \in c(H)$, and $prop_t$ is a function that computes the proportions of the objects in hotspot H belonging to class t, and $\theta > 0$ is the interestingness threshold. Interestingness hotspot whose purity is above θ can be identified using this interestingness function.

4.3 Reward Functions

We grow hotspots from hotspot seeds, which are small regions with high interestingness. While growing hotspots, a *reward function* is employed to assign a reward to interestingness hotspots. The reward function determines the quality of a hotspot based on its interestingness and size. We add one of the neighboring objects to the hotspot in each step of the growing algorithm based on the reward increase calculated by the reward function. In particular, the neighboring object, which increases the reward value most, is added when growing a hotspot. The following reward function is employed in our framework:

$$R(H) = i(H) \times |H|^{\beta} \quad (5)$$

where $i(H)$ is the interestingness of the hotspot, $|H|$ is the size of the hotspot and β is a real number determining the preference for larger regions. In general, we are interested in finding larger hotspots if larger hotspots are at least equally interesting to smaller ones. Consequently, our evaluation scheme uses a parameter ß with $ß > 1$; that is, the reward value increases nonlinearly with hotspot size depending on the value of ß, favoring hotspots with more objects. Selecting larger values for the parameter ß usually results in larger hotspots.

4.4 Neighborhood Definitions

In gridded and polygonal datasets, determining the contiguity of a region is trivial; grid cells or polygons are neighboring if they share an edge. In some cases, it might be desirable to consider polygons or grid cells as neighbors when they only share a point (e.g., diagonal neighbors). However, contiguity or neighborhood relation is not well-defined for point-based datasets, and a neighborhood definition is required to define neighborhood relation between points.

Various neighborhood graphs for point-based datasets have been proposed in the literature. The most popular graphs include Delaunay triangulation, Gabriel graphs, relative neighborhood graphs, Euclidian minimum spanning trees and Beta skeletons. Figure 8 shows comparison of four popular graph types for a dog-shaped dataset.

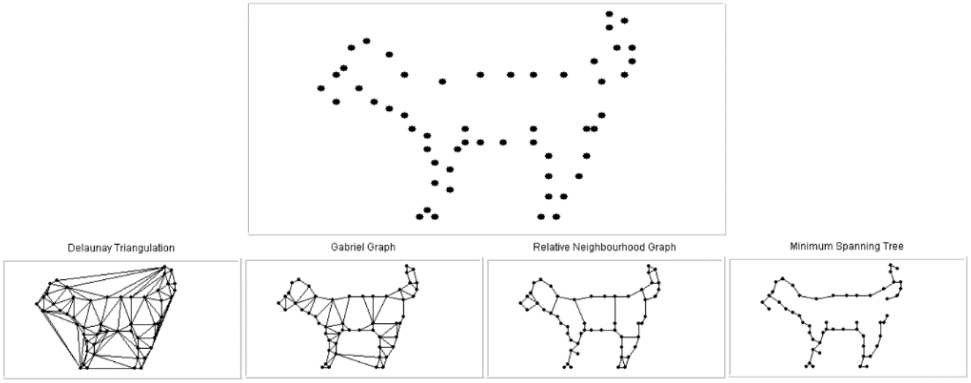


Figure 8. Popular proximity graph types for a dog-shaped dataset

As seen in the figure, the Delaunay triangulation (DT) contains many edges between distant points in the dataset, which are non-intuitive as they capture irrelevant relationships; therefore, it is not a good choice for a neighborhood graph. On the other hand, minimum spanning trees and relative neighborhood graphs contain only a small

amount of connections between points and many close points are not connected, losing important relationships. In contrast, Gabriel graphs strike a good balance; many edges between distant points in the DT are eliminated, yet edges between close points are preserved. Thus, we use Gabriel graphs to identify neighboring objects in spatial datasets. For a more detailed discussion of various neighborhood graphs, we refer to [Matula et al 1980, and Jaromczyk et al. 1992].

CHAPTER 5

METHODOLOGY

In this section, we describe our methodology that works in 5 phases:

(1) Identify neighboring spatial objects

(2) Find small hotspot seeds with high interestingness

(3) Grow the hotspot seeds by adding neighboring objects

(4) Post-process hotspots, typically removing some overlapping hotspots.

(5) Find the scope of each hotspot

Figure 9 depicts the hotspot discovery process and the inputs of each phase. A dataset, an interestingness function and a reward function are the mandatory inputs of the hotspot discovery framework. All inputs shown in orange color are optional and default values are provided that are used when no input for these parameters is given. The output of the hotspot discovery process is a set of hotspots with their scopes.

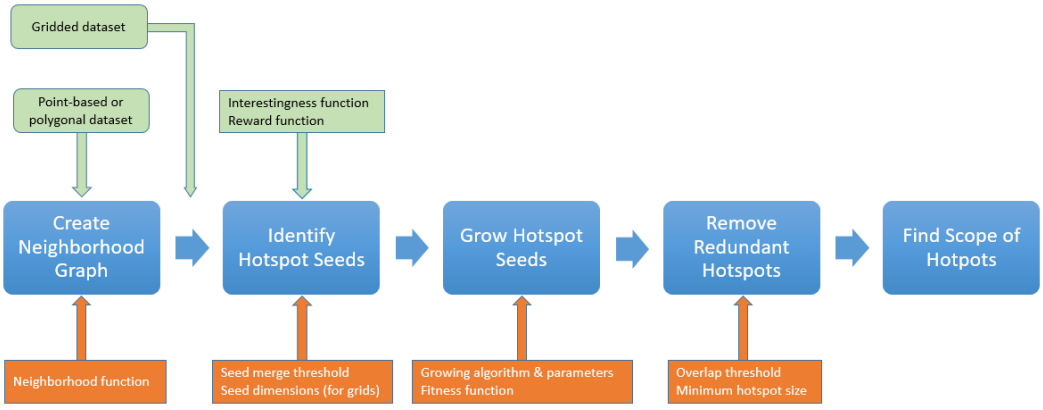


Figure 9. Hotspot discovery process

5.1 Identify Neighboring Spatial Objects:

Our framework supports plugin neighborhood definition functions which take a spatial object as an input and return its neighboring objects. Moreover, our framework provides default neighborhood definitions for point-based, polygonal, and gridded datasets, which will be discussed in this section.

For polygonal datasets, we define two polygons as neighbors if any of their edges are coincident. In our methodology, we calculate the neighboring polygons once and save the neighborhood relations in a neighborhood graph. A neighborhood graph contains a vertex for each polygon, and an edge is created between vertices representing the neighboring polygons. Neighbors for each vertex are stored in an adjacency list.

For gridded datasets, we define two grid cells as neighbors if they share an edge. A neighborhood graph is not required for gridded datasets as finding the neighbors of a grid cell is trivial. Following is the neighborhood definition for a 4-dimensional gridded dataset with x, y, z, and t dimensions:

$$Neighbor(o_1, o_2) \Leftrightarrow |o_1.x - o_2.x| + |o_1.y - o_2.y| + |o_1.z - o_2.z| + |o_1.t - o_2.t| = 1 \quad (6)$$

where o_1 and o_2 are two grid cells and $o_i.x$ corresponds to x dimension value of o_i. Based on this definition, a four-dimensional grid cell has eight neighbors, as the next cell and the previous cell in each dimension is a neighbor.

On the other hand, as discussed in Section 4.4, neighborhood relations between a set of points are more challenging to define. Our framework uses Gabriel graphs to define neighborhoods in a point-based dataset. We introduced Gabriel graphs in

section 2.3.4. In a Gabriel graph, number of edges cannot be more than three times the number of vertices [Matula et al. 1980]. Thus, we assume in our runtime analysis that the number of neighbors for a hotspot region is in the order of O(n) where n is the number of objects in the hotspot.

5.2 Finding Hotspot Seeds

Our methodology depends on identifying hotspot seeds and growing these smaller regions to detect larger hotspots. In this section, we describe how seed regions are constructed for different types of datasets.

Polygonal and point based datasets: Once the neighborhood graph for the dataset has been computed, we create smaller regions around each vertex in the graph that is composed of the vertex itself and its neighbors. We refer to these smaller regions as "hotspot-seed candidates". Then, we calculate the interestingness value for each of these small regions to identify the ones with an interestingness value larger than a predefined "seed interestingness threshold". When a Gabriel graph is used, there can be at most three neighbors per each vertex in average, resulting an average hotspot size of four or less. Seed candidates having an interestingness value larger than the "seed interestingness threshold" are chosen to be grown in the hotspot growing phase. However, we merge many of the neighboring seed regions before growing them, as there is a high degree of overlap between seeds generated by using a neighborhood graph. This merging step is explained in the next subsection.

Gridded datasets: The whole dataset is divided into smaller sub regions of same dimensions and the interestingness value for each of these smaller regions is calculated using the plugin interestingness function. For example, for a four-

dimensional dataset with 10 grid cells in each dimension (with a total of 10^4 grid cells), dividing the dataset into smaller regions of 2×2×2×2 grid cells yields $10^4 / 2^4 = 625$ such regions. Our framework allows user defined seed sizes, therefore the seed candidates can be of any size smaller than the dataset dimensions. Similarly, an interestingness threshold value is used to determine which seed candidates may be used to create larger hotspots. We merge neighboring seeds created for gridded data too, as in most cases neighboring seeds are highly similar and can be merged without losing quality.

5.2.1 Merging Seed Regions

The seeding phase divides the dataset into smaller regions and finds the ones with the higher interestingness. Since we create a hotspot seed candidate around each vertex in the neighborhood graph, many hotspot seeds overlap. Moreover, when we create hotspot seeds for gridded datasets, the *seeds* are disjoint, but we observed that many of these seeds grow to the same (or quite similar) hotspots in the growing phase. This is not surprising, as a large hotspot with high interestingness will usually have smaller sub-regions with very high interestingness. A case study reported in [Akdag et al. 2014] justifies this observation. Apparently, it is inefficient to grow all of the hotspot seeds, so our framework allows merging some seeds before growing them. Moreover, we require that the merge operation produces a new seed with an acceptable reward. That is, we do not want to create hotspot seeds with low rewards as a result of merging two neighboring seed regions. We use a merge threshold μ to determine if the union of seed regions is acceptable, and use the following merge rule: If the reward of the new region is higher than the total reward of merged regions multiplied by μ, then this merge is acceptable:

$$merge(s_1, s_2) \text{ if } R((s_1 \cup s_2)) > (R(s_1) + R(s_2)) * \mu \qquad (7)$$

where $R(s_i)$ represents the reward of seed region s_i, and μ is a real valued parameter. Seed merge threshold μ is usually set to a value larger than 0.9 to make sure that the quality of the new seeds are high. We use the following approach to reduce the number of seeds grown:

(1) Create a neighborhood graph of hotspot seeds, in which there is a vertex for each seed. Create an edge between vertices if the corresponding seeds are neighbors and if the union of these regions produces a new seed with an acceptable reward value according to the merge rule. The weight of each edge is assigned to the reward gain when seeds are merged.

(2) Merge the neighboring seeds that yield the highest reward gain when merged.

(3) Update neighborhood graph after the merge operation: Create an edge connecting the new vertex with the neighbors of the merged vertices using the same procedure (if the union yields an acceptable seed).

(4) Continue merging seed regions as long as there are seeds to be merged (i.e., edges in the graph).

This algorithm is similar to MOSAIC [Choo et al. 2007] clustering algorithm, however is implemented using more efficient data structures, and creates the neighborhood graph using an undirected graph with weighted edges instead of a Gabriel Graph. Algorithm 1 gives the pseudo code for the seed-processing algorithm, and Algorithm 2 gives the pseudo code for the merge procedure which merges seed regions connected by an edge in the graph. Weight of an edge is assigned to the reward gain (line 10 in Algorithm 1) which is calculated by $R((s_i \cup s_j)) - (R(s_i) + R(s_j))$.

ALGORITHM 1. SEED PROCESSING ALGORITHM

```
1:  Create an undirected graph G and hash set S of seed regions
2:  foreach seed region si
3:      Add si to G as a vertex
4:      Add si to S
5:  end foreach
6:  for i = 0 to number of seed regions – 1
7:    for j = i + 1 to number of seed regions
8:        if si and sj are neighbors and R( (si∪sj)) >  (R(si) + R(sj)) * μ  then
9:              Create an edge e connecting nodes si and sj
10:             e.weight = R( (si∪sj))  –  (R(si) + R(sj))
11:             G.AddEdge(e)
12:       end if
13:   end for
14: end for

15: Create a max-Heap H of edges
16: foreach edge e in G.edges
17:     H.enqueue(e, e.weight)
18:  end foreach

19:  while H has elements
20:     nextEdge = H.dequeue()
21:     if S contains both nodes connected by nextEdge then
22:         Merge(nextEdge)
23:     end if
24:  end while
```

We use a hash set data structure to keep a list of seeds in the graph to ensure minimum containment check operation (line 21 in Algorithm 1) time complexity as a hash set data structure has O(1) time complexity for this operation, compared to O(|V|) in a graph data structure. Moreover, we put edges in a priority queue, and use a heap data structure [Cormen 2009] to store the list of edges; as in this problem, we need to prioritize the edges that will be processed. The edge connecting the pair of nodes that results the highest reward gain when merged is the root of the heap and processed first. As discussed in Section 2.2.1, Heap data structure has O(log*n*) time

complexity for extract-max operation, so less time is spent for finding the next merge candidate; and O(log$n$) time complexity for add operation as we add new edges for the new region after a merge operation. Using solely a graph would require O(n) time for extract-max operation.

ALGORITHM 2. SEED MERGE OPERATION

1: Procedure Merge (Edge e)
2: Set s_1 = e.source, s_2 = e.target
3: Merge s_1 and s_2 by adding all elements in s_1 and s_2 in a new region s_{new}
4: Add s_{new} into G as a new vertex
5: Remove e from G
6: foreach neighbor s_i of s_1 connected by edge e_i
7: if $R(s_i \cup s_{new}) > (R(s_i) + R(s_{new})) * \mu$ then
8: Create an edge e_{new} connecting nodes s_i and s_{new}
9: e_{new}.weight = $R(s_i \cup s_{new}) - (R(s_i) + R(s_{new}))$
10: G.AddEdge(e_{new})
11: G.RemoveEdge(e_i)
12: end if
13: end foreach
14: foreach neighbor s_j of s_2 connected by edge e_j
15: if $R(s_j \cup s_{new}) > (R(s_j) + R(s_{new})) * \mu$ then
16: Create an edge e_{new} connecting nodes s_j and s_{new}
17: e_{new}.weight = $R(s_j \cup s_{new}) - (R(s_j) + R(s_{new}))$
18: G.AddEdge(e_{new})
19: G.RemoveEdge(e_j)
20: end if
21: end foreach
22: Remove s_1 and s_2 from the graph G and hash set S
23: Add s_{new} into hash set S
24: End procedure

While processing edges in the order of descending weights, it is possible that an edge which is still in the heap might have been removed from the graph by the merge procedure as a result of merging one of the connected nodes in a previous step. That edge might still be in the heap as the heap data structure does not support deleting a particular node. In that case, the dequeue operation will return an edge that does not actually exist in the graph. So, we check if both nodes connected by the edge exist in the set of seed regions to make sure that both nodes survive (line 21 in Algorithm 1); otherwise we just skip processing that edge.

5.3 Growing Hotspot Seeds

In this phase, we grow hotspot seeds by adding neighboring objects. We will firstly present a trivial baseline algorithm and then we will improve the algorithm by using a more efficient data structure.

5.3.1 Baseline Hotspot Growing Algorithm:

The baseline algorithm grows hotspots by finding the neighboring spatial object in each step which will increase the reward function the most when added to the region. This neighbor is then added into the region and region's neighbors list is updated with the neighbors of the newly added object which are not already a neighbor or already in the region. We save the best hotspot obtained so far, and update the best hotspot when a "hotter" hotspot has been found. We continue adding neighbors as long as the hotspot's interestingness remains positive. Figure 10 gives an example of how a hotspot's reward value changes while growing. This hotspot was created by running hotspot growing algorithm on a real gridded dataset. The hotspot starts growing with 9 grid cells and

the reward increases until it reaches a local maxima at 72 grid cells with a reward value of 28 and then the reward starts to decrease. However, after adding a few more neighboring grid cells to the hotspot, the reward value starts to increase again and reaches the global maximum value of 37.2 at 131 grid cells. Thus, it is essential to continue growing hotspots even after the reward value starts decreasing.

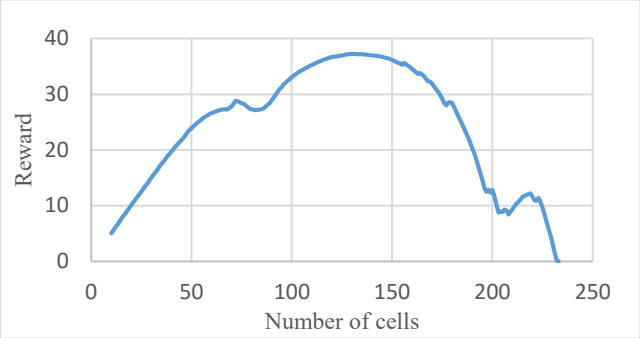


Figure 10. Change of reward value of a growing a hotspot

We keep the objects in the region, and objects in the neighbor's list in two separate hash sets as hash set data structure has optimal O(1) runtime complexity for Add, Remove and Contains operations which are used extensively while growing a hotspot. Computational complexity of the hotspot growing phase is $O(|H|) \times O(R(|H|))$ for each iteration of a hotspot where $|H|$ is the cardinality of a hotspot and $O(R(|H|))$ is the runtime complexity of calculating the reward value of hotspot H. In each iteration, reward value with each neighbor is calculated. Assuming that the number of neighbors is in the order of the size of the hotspot, as the number of objects increase in each iteration, a hotspot grows from h_0 initial elements to n elements, and if $O(R(|H|)) = O(|H|)$, the runtime complexity of growing a hotspot is:

$$O\left(\sum_{h=h_o}^{n} h^2\right) = O(n^3)$$

If the reward is calculated incrementally in O(1) time, then the complexity is reduced to:

$$O\left(\sum_{h=h_o}^{n} h\right) = O(n^2)$$

In the next section, we will demonstrate how to calculate variance interestingness incrementally.

5.3.2 Incremental Calculation of Interestingness Functions

Incremental calculation means that as new data arrives, a function's output is updated without going over the previous data. For example, count or sum of elements in a dataset can be calculated incrementally by just increasing the value of a variable, instead of counting or adding all numbers again. However, not all functions can be calculated incrementally. Aggregate functions are categorized as distributive, algebraic or holistic functions [Shekhar et al. 2003]. Distributive functions can be computed in a distributive manner by partitioning the data into subsets, and aggregating the results of applying the function to each subset; therefore, it is trivial to calculate distributive functions incrementally by considering each new data as another subset, and aggregating with the previous value. Sum, count, min, max are examples to distributive functions. Algebraic functions can be computed by applying an algebraic function on a constant number of distributive functions. Therefore, algebraic functions can also be calculated incrementally. For example, average function is an algebraic function as it can be computed by sum/count. On the other hand, holistic functions cannot be computed by applying an algebraic function on a constant number of distributive functions, all data needs to be processed together. As

a result, unlike distributive and algebraic functions, holistic functions cannot be calculated incrementally. Median and rank are examples of holistic functions.

In our framework, we calculate variance, correlation and purity interestingness functions incrementally as they are algebraic functions. In this subsection, we describe a methodology for implementing a given algebraic or distributive function incrementally, using variance function as an example.

Given an empty set of objects S, we are supposed to calculate the variance of attribute a incrementally while adding new objects into S where each object has an attribute a. It was shown [Welford 1962] that variance can be calculated incrementally by updating the mean and sum of squared differences (M_2) with each new value x using the following equations:

$$mean_n = mean_n\text{-}1 + (x - mean_n\text{-}1) / n \text{ , and}$$

$$M_2 = M_2 + (x - mean_n\text{-}1) \times (x - mean_n)$$

where $mean_n$ is the mean of n numbers. Then the variance can be calculated in O(1) time by:

$$variance(S, a) = M_2 / (n\text{-}1)$$

where n is the number of objects in the dataset. We use the same equation in our implementation to calculate variance in O(1) time when a new object is added to a region. From a software design perspective, we create a new class named "StatsCalculator" which keeps a reference to n, mean and M_2. Each hotspot has its own copy of StatsCalculator object. Each time we add a new object to a hotspot, we call the procedure "Add(object)" in StatsCalculator class, which updates these values with the

value of attribute a of the object. When we need to retrieve the variance value for the hotspot, we call "variance()" procedure which just returns $M_2/ (n-1)$.

Calculating correlation function incrementally is similar but more complicated. We will not present the details and refer to the literature [Pebay 2008].

5.3.3 Heap-based Hotspot Growing Algorithm

In this subsection, we introduce a novel heap-based hotspot growing algorithm. In the hotspot growing phase, the baseline algorithm searches for the best neighbor among all neighbors, and after each time a new neighbor is added this search is repeated. For example, assuming that reward function is calculated incrementally, searching for the best neighbor in each step takes the complexity of hotspot growing algorithm to $O(n^2)$ where n is the number of objects in the hotspot (in all analysis we assume there are $O(|H|)$ neighbors for a hotspot H of size $|H|$). However, we observed that the ordering of the neighbors according to their 'fitness' for the region does not change much as the region grows. If a neighbor n_1 increases the reward more than the neighbor n_2 when evaluated in step s_i, this *mostly* means that n_1 is still a better fit to be included in the region in step s_j (which means that n_1 has a better fitness for the hotpot). There are some cases this does not hold: if n_1 and n_2 have very close fitness values, as the region changes while growing, n_2 may become a better fit for the region. However, by experimental evaluations we observed that such cases occur very rarely and do not affect the final hotspot significantly, as both neighbors are generally either included in or excluded from the hotspot. Moreover, if the interestingness function is defined on an attribute of objects (rather than being defined on a set of objects, like correlation and variance), then that attribute directly determines the fitness of an object. Thus,

it is mostly unnecessary to evaluate each neighbor in each step of the growing phase. Instead, we use a max-heap data structure to keep the list of neighbors where the neighbor with the highest fitness value is the root of the heap tree. Using a max-heap, instead of searching for the best neighbor in each step, we simply add the root node into the region. When new neighbors are encountered while growing the hotspot, we assign each new neighbor a *fitness value* by either evaluating the reward gain in case the neighbor is added to the region, or by considering the interestingness attribute and add it into the heap using the *fitness value* as the priority of the new node. Next, we continue growing the region as long as there are more neighbors in the heap and the interestingness of the region is higher than the interestingness threshold.

Heap data structure has O(log$n$) time complexity for extracting the root node (extract-max operation), insertion and deletion operations. Some specialized heap implementations (Binomial heaps and Fibonacci heaps [Cormen 2009]) allow amortized O(1) time complexity for insertion. Algorithm 3 gives the pseudo code of heap-based hotspot growing algorithm. This procedure is called repeatedly in GrowRegion function (line 16) as long as the interestingness of the region remains positive and there are more neighbors in the heap. The best reward found is memorized (lines 11-14) and when hotspot growing is finished, the hotspot is set back to this state. The runtime complexity of the heap-based hotspot growing algorithm is O($n$log$n$) as a total of O(log$n$) time is spent in each step where n is the number of objects in the hotspot—O(log$n$) time is spent for finding the best neighbor and removing it from neighbors list. In addition, O(1) time is spent for adding it to the region and O(log$n$) time is spent for adding neighbors of newly added object to the neighbors list. Before adding the neighbors of the newly added object to the heap, it needs to be ensured that

the neighbor is not contained in the region or in the neighbors list. To optimize the time complexity for this containment check, we keep the objects in the region in a hash set. Furthermore, in the implementation of max-heap data structure, we put all elements in the heap into an additional hash set data structure and manage them together to optimize the runtime complexity when checking if an object is in the neighbor's list (which would be O(*n*) using only the heap.)

ALGORITHM 3. HEAP-BASED HOTSPOT GROWING ALGORITHM

```
1:  Procedure AddNextNeighbor(region)
2:      set bestNeighbor = Heap.dequeue()
3:      add bestNeighbor to region
4:      set newReward = CalculateReward(region)
5:      foreach neighbor n of bestNeighbor
6:          if n is not in region and n is not in the neighbors list then
7:              set fitness = CalculateFitness(region, n)
8:              Heap.enqueue(n,fitness)
9:          end if
10:     end foreach
11:     if newReward > region.alltimeBestReward then
13:         set region.alltimeBestReward = newReward
12:         set region's alltimeBestObjects = region.objects
14:     end if
15: end procedure
16: Procedure GrowRegion(region)
17:     while region's interestingness is positive and region has neighbors then
18:         AddNextNeighbor(region)
19:     end while
20: End Procedure
```

Using a fitness value to order neighbors requires choosing a good fitness measure for each neighbor of the hotspot. New neighbors are evaluated with a different state of the growing hotspot. Thus, it is very important to put the new neighbors in a correct order in the max-heap. If the interestingness value of a spatial object is based on an

attribute of the object, then this attribute value can be directly used as its fitness value. If a more complicated interestingness function is used, which is calculated on a set of objects, then the reward increase and hotspot size can be used to define a fitness value for the neighbors. Sample fitness functions are given in the experimental evaluation section. Heap-based growing algorithm is a greedy algorithm, as it makes a locally optimal choice in each step. The ordering of the neighbors based on their fitness may not be optimal in each step; however, this algorithm is much more efficient than the baseline algorithm.

Furthermore, we grow each hotspot seed in parallel, as growing one seed does not affect growing other seeds. We use a shared memory parallel programming approach.

5.3.4 Dimensional Growing Algorithm for Gridded Datasets

In this subsection, we propose an alternative hotspot growing algorithm for gridded datasets which grows seed regions by extending them in the dimension that increases the reward value most, in each step. If the region is two-dimensional, this process is similar to increasing the length or width of the rectangular seed region in each step based on which one increases the reward value most. For a rectangular hotspot with x and y coordinates, as seen in Figure 11, there are 4 possible ways to extend: Increasing x on east direction, increasing x on west direction, increasing y on north direction and increasing y on south direction. On the other hand, for a 3D hotspot there are six different ways to extend as z coordinate can also be extended. If we think of this hotspot as a rectangular prism (Figure 12), we are making it either taller, larger or wider in each step. Similarly, for a 4D hotspot there are eight directions for extension. Our framework allows domain users to grow hotspots on a subset of dimensions.

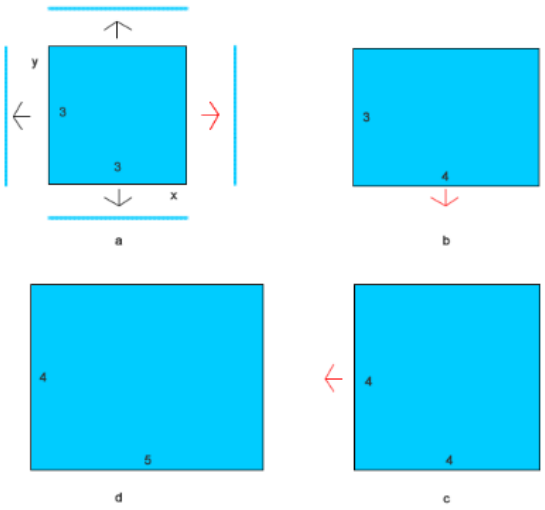


Figure 11. Dimensional growing of a 2D region

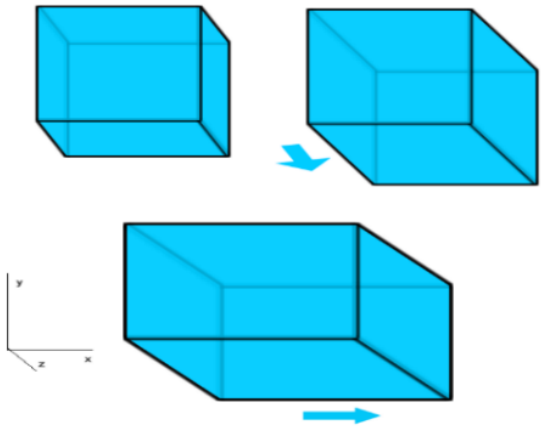


Figure 12. Dimensional growing of a 3D region

Algorithm 4 gives the pseudo code for the dimensional growing algorithm. We continue extending the hotspot as long as the hotspot's interestingness is positive. We keep a reference to the best reward value found so far and we output the state of the hotspot with the highest reward value as the result.

ALGORITHM 4. DIMENSIONAL GROWING ALGORITHM

1. While seed region can be extended and its interestingness value is positive:

2. Find the direction that improves the reward value most when the region is extended in this direction

3. Extend the region in the direction chosen

4. Update region's interestingness and reward value

5. If new reward is larger than the best reward found so far, update the best reward found so far and memorize the state of the region

6. Output the state of the region with the best reward

Computational complexity of the dimensional hotspot growing phase is $O(|H|^{(d-1)/d} \times d \times 2) \times O(R(|H|)$ for each iteration of a hotspot where $|H|$ is the cardinality of an hotspot, d is the number of dimensions and $O(R(|H|)$ is the runtime complexity of calculating the reward value. In each iteration, the new reward value is calculated for growing dimensions in all directions and there are d×2 directions. Assuming that the region has same size in each dimension, on average there are $|H|^{(d-1)/d}$ grid cells in each direction. We add this number of grid cells for each direction in each step. Thus each step requires $O(|H|^{(d-1)/d}) \times d \times 2 \times O(R(|H|)$ operations in each step. Since region's size is increased by a factor of $|H|^{(d-1)/d}$ in each iteration, the number of steps required to reach n cells is significantly lower than heap-based growing algorithm. The runtime complexity depends on number of dimensions and hotspot's shape, however it is definitely much lower than the complexity of heap-based growing algorithm and very

close to O(|H|) when reward is calculated incrementally as shown in the experimental evaluation section.

5.4 Post-processing: Finding an Optimal Set of Hotspots

In general, hotspot growing algorithms create numerous hotspots with a high degree of overlaps. The goal of this phase is to reduce this highly redundant set to a set of high-quality hotspots with a low degree of overlaps and eliminate the overlapping ones with low quality. To do this, we set an overlap threshold and find an optimal set of hotspots that overlap less than the threshold value while maximizing the total reward. More formally, the post-processing problem can be defined as follows:

Input: a set of hotspots S, and an overlap threshold λ where $0 \leq \lambda < 1$,

Problem: Find a subset $S' \subseteq S$ for which $\sum H \in S$ Reward(H) is maximal

where Reward(H) is the reward of hotspot H, subject to the following constraints: $\forall H_1 \in S'$ and $\forall H_2 \in S'$ $(H1 \neq H2 \Rightarrow overlap(H_1, H_2) \leq \lambda)$

where

$$overlap(H_1, H_2) = \frac{|H_1 \cap H_2|}{\min(|H_1|, |H_2|)} \quad (8)$$

Our framework uses the reward of each hotspot as the value of a hotspot; thus, we try to maximize the total reward value of the resulting hotspots.

We define the degree of overlap (8) between two hotspots as the ratio of the number of objects that are shared between both hotspots to the number of objects in the hotspot with the smaller size. For example, if one hotspot has 100 objects, and another one has 80 objects and they share 60 objects, then the degree of overlap is 60/80 = 0.75. In this

definition, the number of objects in the smaller hotspot is used in the denominator to ensure that small hotspots contained in larger hotspots are eliminated. Alternatively, the total number of objects in both hotspots could be the denominator, however, if hotspot A with 1000 objects completely contains all objects in the hotspot B which has 100 objects, then the overlap ratio would be 100/1100 = 0.09, which would imply a very low degree of overlap. Definition (8) solves this problem. However, our framework is quite extensible and allows using plugin definitions for overlap function.

In the remainder of this section, we formulate the optimization problem introduced in this section as a graph problem in which the goal is finding the independent set of vertices in the graph with the maximum total weight and present a methodology that finds the optimal solution in multiple steps. The steps of our methodology for removing redundant overlapping hotspots are as follows:

(1) Calculate the reward of each hotspot using the plugin reward function or use the existing reward values calculated.
(2) Create a weighted overlap graph of hotspots in which weight of each vertex is the reward of the hotspot and there is an edge between two vertices if their degree of overlap is more than the overlap threshold λ.
(3) Simplify the overlap graph by eliminating vertices that cannot be in the optimal solution.
(4) Find the connected components in the simplified overlap graph.
(5) For each connected component C_i, create the complement graph $C_{i'}$.
(6) Find the maximum weight clique (MWC) in each complement graph $C_{i'}$.
(7) The union of all vertices in MWCs is the optimal solution

5.4.1 Creating an Overlap Graph

In this subsection, we present the methodology for creating an overlap graph for a set of hotspots.

Definition 5.1 (Overlap Graph). A hotspot *overlap graph* is a weighted undirected graph G(V,E) in which each vertex v corresponds to a hotspot and weight of each vertex corresponds to the reward of the hotspot. There is an edge between two vertices v and u if and only if the degree of overlap between hotspots represented by v and u is larger than the overlap threshold λ. The weight of each edge is based on the degree of overlap between the corresponding hotspots.

When creating the overlap graph, the reward of each hotspot calculated by the hotspot growing algorithm is used as the weight of each vertex in our methodology. However, this is not a strict requirement as our framework allows using a plugin reward function for post-processing step. We calculate the degree of overlap between each pair of hotspots using the plugin overlap function and create an edge in the overlap graph between vertices representing these hotspots if the degree of overlap is more than the overlap threshold. Figure 13a shows a sample set of overlapping hotspots with their reward values. Figure 13b shows the overlap graph in which there is an edge (solid lines) between vertices if the degree of overlap between hotspots is larger than 0.4 according to definition 8, which requires that one hotspot needs to contain 40% of another to be considered as overlapping. Dashed lines in Figure 13b represents overlapping hotspots; however, the degree of overlap was less than the overlap threshold, so those edges were removed in the final graph (Figure 13c).

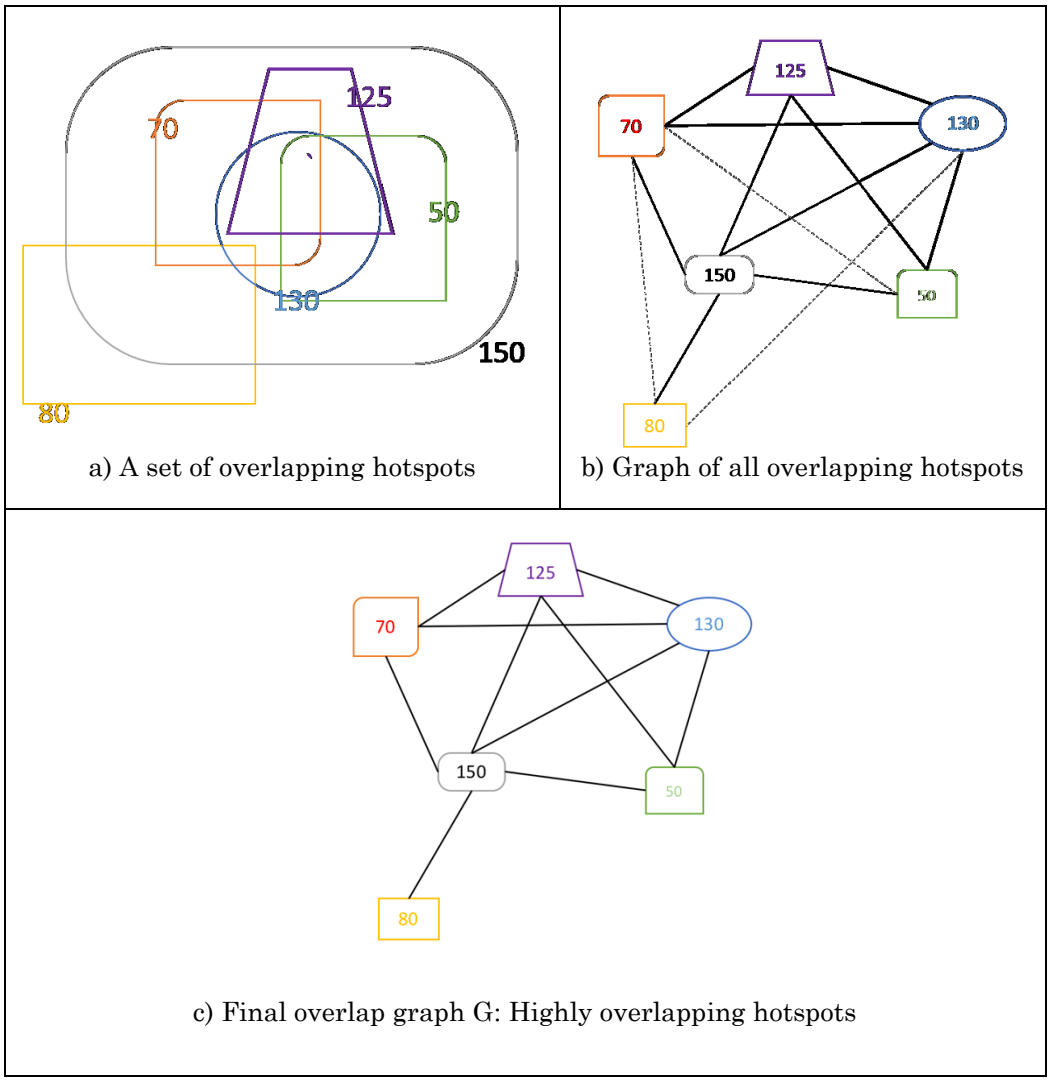


Figure 13. Creating an overlap graph a) A sample set of overlapping hotspots b) Graph of all overlapping hotspots c) Final overlap graph G

Since the goal is to find an optimal set of non-overlapping hotspots that maximize the total reward, and an edge between vertices indicates overlap, this optimization problem is now reduced to finding a set of vertices in G maximizing the total weight where there is no edge between any pair of vertices. As defined in Section 2.1, an independent set is a set of vertices in a graph in which there is no edge between any pair of vertices; therefore, we reduce this problem to finding the maximum weight

independent set in the overlap graph. Considering the graph in Figure 13c, the set of vertices with weights {70, 50, 80} are all independent, thus this set is an independent set. Some of the other independent sets include {125, 80}, {130, 80}, {70, 50} and each vertex is also an independent set by itself.

We can also think of this problem in the following sense, considering the complement graph of G: If there is no edge between a pair of vertices in G, which means they are independent, then there will be an edge between those vertices in the complement graph G'. The problem can now be converted to finding the subset of vertices in G' with the maximum weight in which there is an edge between all pairs of vertices, which is the maximum weight clique. Considering the graph in Figure 13c, the set of vertices with weights {125, 150, 50}, {70,150, 125}, {80, 150}, {130} are some of the cliques. Each vertex itself is also a clique.

The problem of finding the maximum weight clique (MWC) or its dual problem of finding the maximum weight independent set in a graph are well-known NP-hard problems and there has been exhaustive research on this topic. Bomze et al. [1999] gives a survey of exact and approximate solutions to this problem. This problem is also known to be a hard-to-approximate NP-hard problem as shown by [Gary and Johnson 1979]. That is, the optimal solution cannot be efficiently approximated to a certain degree. In our methodology, we use the maximum weight clique algorithm proposed by Östergård [2002], which finds the optimum solution, moreover its implementation is available for public use and it is quite fast—it can find maximum weight cliques in graphs with up to a few hundred vertices often under a second. However, it takes hours when the input graph is too complex with thousands of vertices; thus, we preprocess

the overlap graph and significantly simplify and partition the graph to improve the efficiency of the methodology which is the subject of the next section.

5.4.2 Simplification of Overlap Graph

We simplify the graph by removing the vertices which are guaranteed to be eliminated by maximum weight independent set or maximum weight clique algorithms. We define "overlap set" of a hotspot as the set of hotspots the hotspot overlaps with, including the hotspot itself. It is obvious that if two hotspots overlap and if they overlap with the same set of hotspots, then their overlap sets will be same, and the one with the higher reward will always be chosen in the maximum weight independent set in case one of these hotspots will be in the in this set at all. This is also true for a set of hotspots, that is, in a set of overlapping hotspots, if they all have the same overlap sets, only one of them can be selected into the maximum weight independent set and this will be the hotspot with the highest reward. Keeping only the best hotspot among such hotspots reduces the graph size dramatically and improves the efficiency of the framework significantly.

Algorithm 5 depicts the algorithm we use for simplifying the graph. The following steps summarizes the simplification algorithm:

(1) For each vertex, create a Set data structure and put the vertex itself and all of its adjacent vertices into the set. This set will be called "overlap set" of a vertex.

(2) Compare each vertex's overlap set with the overlap set of other vertices with which this vertex is connected. Add a vertex into a "removal set" if its weight is lower than a vertex with the same overlap set.

(3) Remove vertices in the removal set from the overlap graph G.

ALGORITHM 5. OVERLAP GRAPH SIMPLIFICATION CODE

```
1. Procedure SimplifyGraph(G)
2.         foreach vertex vi in G
3.               si  = overlap set of vi
4.         end foreach
5.         Set RemovalSet =  Empty set of vertices
6.         foreach vertex v1 in G
7.            if v1 is in RemovalSet then continue;
8.            foreach vertex v2 adjacent to v1
9.               if v2 is in RemovalSet then continue;
10.                 if |s1| = |s2| and |s1 ∪ s2| = |s1| then
11.                    if v1.weight < v2.weight then
12.                       RemovalSet.Add(v1)
13.                    else
14.                       RemovalSet.Add(v2)
15.                 end if
16.           end foreach
17.        end foreach
18.        foreach  vertex vj in RemovalSet
19.             G.Remove(vj)
20.        end foreach
21. End Procedure
```

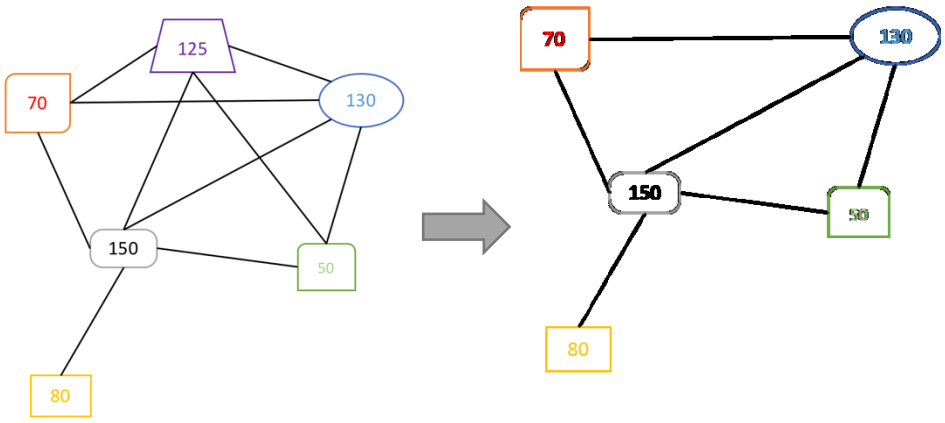


Figure 14. Simplified overlap graph (on the right)

Figure 14 visualizes the overlap graph before and after the simplification. In the simplification process, firstly an overlap set is created for each vertex. The vertex with weight 125 in Figure 13c has an overlap set of {50,70,125,130,150} which is same as

the overlap set of vertex with weight 130; thus, it is impossible for this vertex (125) to be in the maximum weight independent set as its weight is smaller than 130. The overlap sets of all other pair of vertices are different, so no more simplification is done on this graph. In graphs with a very large number of overlaps, simplification step significantly simplifies the graph.

Assuming that each hotspot has a constant set of overlapping hotspots, the worst case runtime complexity of the simplification algorithm is $O(|V|)$ where $|V|$ is the number of vertices (hotspots) in the overlap graph. Without this assumption, the runtime would be $O(|V|^3)$ in case all hotspots are overlapping with most others due to lines 6-10 in Algorithm 5. In our implementation of the set data structure, we use a hash set, which assigns a hash value to each set element; therefore, checking for inclusion of an element in the set, adding/removing an element to/from the set is all achieved in O(1) time.

To the best of our knowledge, simplification of the graph while calculating the maximum weight clique or independent set using an overlap set is unique to our approach. We claim that the proposed simplification algorithm can be employed as a preprocessing step by algorithms that find maximum weight cliques and maximum weight independent sets in graphs with highly overlapping vertices. Next, we do another optimization by partitioning the graph into sub-graphs.

5.4.3 Partitioning the Overlap Graph

In this subsection, we show how we partition the overlap graph into sub graphs. By definition, vertices in each connected component of a graph are independent from vertices in other connected components, which allows further optimizations for finding

the maximum weight independent set in the overlap graph. Instead of running an NP-hard algorithm on the whole overlap graph, it makes sense to run the algorithm for each connected component which were already simplified. We identify the connected components in the overlap graph G and then find the maximum weight independent set of each connected component C_i by finding the maximum weight clique for the C_i' (complement of C_i). The final optimal solution is the union of all vertices in the "maximum weight independent set" of all connected components.

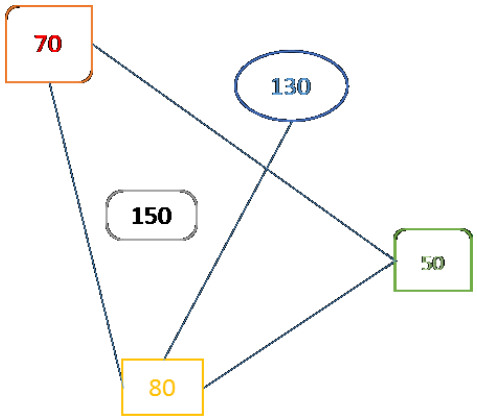


Figure 15. Complement of the graph

There were only one connected component in the graph in Figure 14. Therefore, we create the complement graph for this component (shown in Figure 15) and find the maximum weight clique in it. In this example, there are many possible cliques. Each vertex itself is a clique. Vertices with weights 70, 50 and 80 create a clique of size 3 with a total reward of 200—they are not adjacent in Figure 13, and they are all pairwise adjacent in Figure 15. On the other hand, vertices with weights 130 and 80 create a clique of size 2 with a total weight of 210 and this subset yields the maximum weight clique.

5.4.4 Summary

We claim that the proposed hotspot post-processing algorithm can be used with our algorithms that create overlapping hotspots or clusters. Most hotspot discovery algorithms (such as SatScan) do not employ reward functions, in this case, a reward value to each hotspot can be assigned depending on the methodology used, or based on a user defined function which may consider hotspot size and other internal or external evaluation measures. SatScan assigns a likelihood ratio to each hotspot, which can be used as the reward value of hotspots. Moreover, algorithms for finding the maximum weight cliques and maximum weight independent sets can employ the proposed graph simplification algorithm as a pre-processing step to significantly simplify the input graphs and improve their runtimes. The graph simplification algorithm is quite efficient and it takes under a second to simplify graphs with hundreds of vertices and tens of thousands of edges.

5.5 Finding the Scope of Hotspots

In this subsection, we discuss how to compute the scope of hotspots. For polygonal datasets, determining the scope of the hotspots is trivial: Merging all polygons that form the hotspot will create a large polygon—the obtained polygon is the scope of the hotspot. The same procedure is also applied to the gridded datasets: the boundaries of the union of all grid cells will create a polygonal border for the gridded datasets in any dimensions. However, it is not trivial to compute the scope for a point-based hotspot. Many possible boundaries can be defined for a set of points and finding a representative boundary is a tricky task. In this section, we present a methodology for creating polygon models for 2-dimensional point-based hotspots.

5.5.1 Polygon Models for Clusters

Polygons serve an important role in the analysis of spatial data. In particular, polygons can be used as a higher order representation for spatial clusters, such as for defining the habitat of a particular type of animal, for describing the location of a military convoy consisting of a set of vehicles, or for defining the boundaries between neighborhoods of a city consisting of sets of buildings. As the existing clustering algorithms return clusters represented as a set of points and not as a model, it is attractive to use polygons as cluster models due to the following reasons.

First, it is computationally much cheaper to perform certain calculations on polygons than on sets of objects. For example, Cao et al. [2013] uses polygons have been used to describe the functional regions of a city. A given location can be assigned to one of those functional regions very efficiently by checking in which polygon the location is included.

Second, relationships and changes between spatial clusters can be studied more efficiently and quantitatively by representing each spatial cluster as a polygon. Polygon analysis is particularly useful to mine relationships between multiple related datasets, as it provides a useful tool to analyze discrepancies, progression, change, and emergent events [Wang et al. 2013].

However, there is not an established procedure in the literature on how to derive polygonal models from spatial clusters. The objective of this section is to find an optimal set of polygons for two-dimensional spatial clusters and hotspots. All hotspots can be considered as a cluster. Thus, we will use the term cluster in this section to refer to hotspots and clusters in two-dimensional space.

The input of this process is a spatial cluster containing a set of points and its output is a set of polygons—the model of the cluster. However, it is not trivial to generate such representative polygons. As shown in Figure 16, many different polygon models (or a set of polygons as in Figure 16e) can be generated for the same set of points. Therefore, it is desirable to define application specific criteria for evaluating different polygon models. Depending on the application context, a different one of the seven shapes in Figure 16 may be desirable.

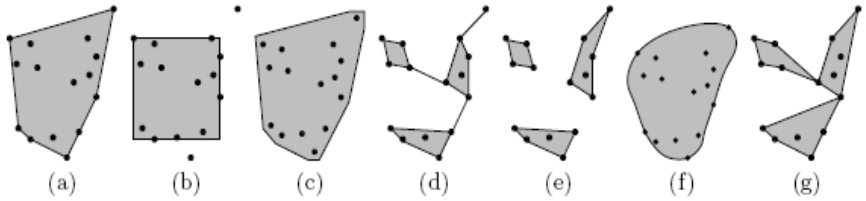


Figure 16. Different shapes generated for the same set of points

5.5.2 Existing Methods for Creating Polygon Models

Representing a set of points as polygons (or similar geometric shapes), creating the boundary of a set of points, or defining the perceived shape of a dot pattern has been a research area in computational geometry, computer graphics, computer vision, pattern recognition, and geographic information science for many years.

Convex hulls are the simplest way to enclose a set of points in a convex polygon. However, convex hulls may contain large empty areas that are not desirable for good representative polygons. Creating polygon models based on Voronoi diagrams or Delaunay triangulations is another commonly used approach. Alani et al. [2001] describe a method for generating approximate regional extents for sets of points that are respectively inside and external to a region. However, the proposed method

requires defining a set of points outside the given cluster so that cluster boundaries can be obtained. Matt Duckham et al. [2008] propose a "simple, flexible, and efficient algorithm for constructing a possibly non-convex, simple polygon that characterizes the shape of a set of input points in the plane, termed a Characteristic shape". The algorithm firstly creates the Delaunay triangulation of the point set—which actually is the convex hull of the point set—and then reduces it to a non-convex hull by replacing the longest outside edges of the current polygons by inner edges of the Delaunay triangulation until a termination condition is met.

The Alpha shapes algorithm, introduced by Edelsbrunner et al. [1983] also uses Delaunay triangulation as the starting step and generates a hull of polylines, enclosing the point set and this hull is not necessarily a closed polygon. Thus, the Alpha shapes algorithm requires post-processing for creating polygons out of the polylines. Besides, there is no easy way of determining the proper parameter for Alpha shapes algorithm.

A. Ray Chaudhuri et al. [1997] introduce s-shapes and r-shapes; the proposed algorithm firstly generates a staircase like shape called s-shape, which is determined using an s parameter and then reduces it to a smoother shape using the r parameter. The algorithm can cope well with varying densities in the point set. However, there is no easy way to estimate a good r parameter; the authors state that "to get a perceptually acceptable shape, a suitable value of r should be chosen, and there is no closed form solution to this problem".

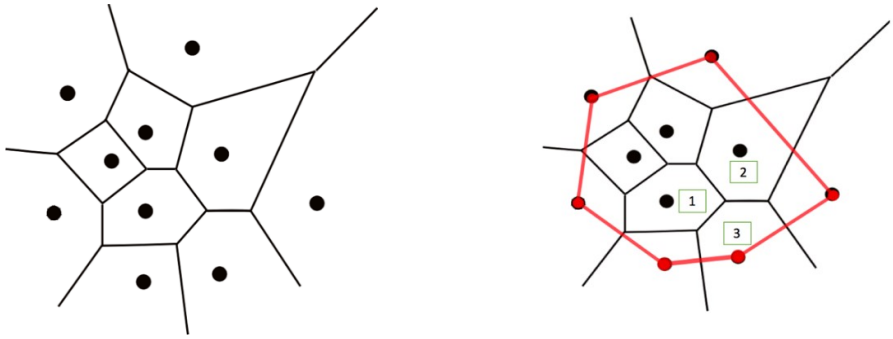
A commercial algorithm, called Concave Hull [Moreira and Santos 2007], generates polygons by using a method that is similar to the "gift-wrapping algorithm" used for generating convex hulls. It employs a k- nearest neighbors approach to find the next point in the polygon and creates a simple connected polygon unless the smoothness

parameter k is too large and the points are not collinear. A density-based clustering algorithm, DContour [Chen et al. 2009] is the only algorithm, which is known to use density contouring for generating polygonal boundaries for a point set. However, when using this approach selecting the proper kernel width for the density estimation approach is non-trivial.

5.5.3 A Voronoi Diagram Based Method for Finding Scopes of Hotspots

As discussed in the previous subsection, Voronoi diagrams can be used to create polygonal boundaries for point-based datasets when a set of points outside the region is given, so that the unbounded Voronoi edges can be bounded with a polygon boundary.

Our approach uses polygon models and computes the scope of point-based hotspots operating on the Voronoi diagram and the convex hull for the spatial dataset. We firstly calculate the Voronoi diagram and convex hull of the whole dataset. Either each point in a hotspot will be in a Voronoi polygon, or if the point is on the convex hull of the dataset, it will not be enclosed by a Voronoi polygon—in which case it will be in an unbounded Voronoi region. In this case, we propose enclosing such points in a polygon by intersecting the convex hull of the dataset with the unbounded Voronoi regions. Moreover, some points will not lie on the convex hull, but they will be enclosed by a polygon, which crosses the convex hull. Such points and their Voronoi cells usually lie on the boundary of the dataset and their Voronoi polygons are quite large, beyond the convex hull. To avoid this, we intersect such Voronoi polygons with the convex hull to obtain hotspots that are more compact.



a) Voronoi cells for a set of points b) Convex hull for the same set of points

Figure 17. Voronoi diagram and convex hull for a set of points

Once the Voronoi diagram and the convex hull of the dataset is created, we propose the following algorithm for creating a polygon model for a spatial hotspot:

1. Create an initial empty polygon set PS for the hotspot.
2. For each point P in the hotspot:
    a. If P is in a closed Voronoi cell (Voronoi polygon), check if it crosses with the convex hull:
        i. If the convex hull does not cross a Voronoi polygon, then add this polygon to PS. (Region labeled 1 in Figure 17b)
        ii. If the convex hull crosses the Voronoi polygon, then the convex hull splits this polygon into two polygons. In this case, the point will be inside one of these polygons. Add the polygon to PS. (Region labeled 2 in Figure 17b)
    b. If the point is not in a Voronoi polygon: find the intersection of the Voronoi edges around the point and the convex hull. The intersection will create a polygon; add this polygon into PS. (Region labeled 3 in Figure 17b)
3. Return $P=\cup_{p\in P}$ as the scope of the hotspot.

The result of merging the three numbered regions in Figure 17b would be the polygonal shape colored in pink in Figure 18. As seen in the figure, our approach bounds the unbounded Voronoi regions by using the Convex Hull, creating a compact polygon.

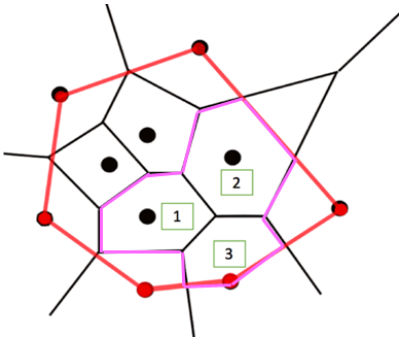


Figure 18. Polygon model for the region in Figure 17b

This method creates polygons for all points in the hotspot and merges them. Since all points in the hotspot are connected, the union of all polygons creates one large polygon model for the hotspot.

On the other hand, Voronoi diagram divides the area between a point and its neighbors evenly. However, in some cases it might be desirable to create tighter boundaries excluding the area between different hotspots. Moreover, the hotspot locations might be given as an output from a hotspot discovery algorithm, and the whole dataset might not be available. We provide another methodology for creating polygon models for hotpots, and clusters in general, to address these concerns. In the next sections, we will discuss the desired polygon models in such cases, and describe a methodology for creating desired polygon models.

5.5.4 Desired Polygon Models

In this subsection, we discuss desired polygon models for clusters. Figure 19 depicts three polygons that were created for the same cluster:

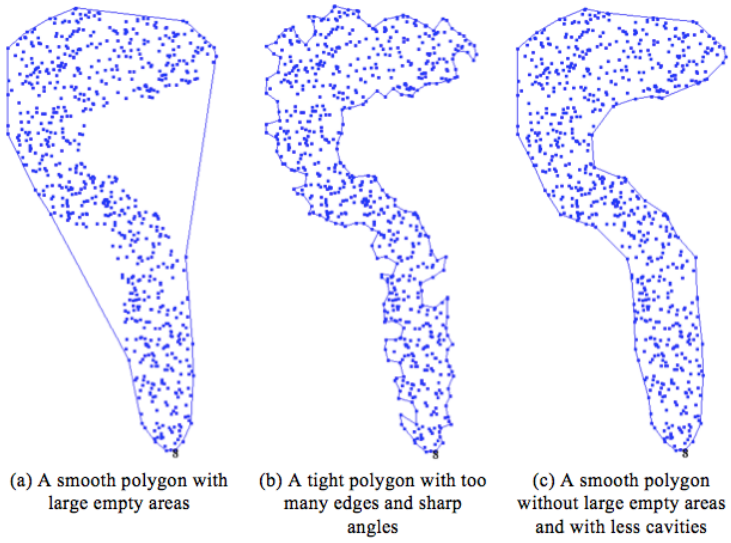


Figure 19. Polygon models generated for a cluster

The generated polygon in Figure 19a covers the largest area, and has the smallest perimeter, the least number of edges and the smoothest shape. However, it is obviously not a good model for the cluster because it includes large empty areas that are not relevant to the cluster. On the other hand, the polygon in Figure 19b has the largest perimeter, the most number of edges and covers the smallest area. Yet, it is also not a good representation for the cluster due to its ruggedness. Additionally, this polygon has a potential overfitting problem, as it is quite complex and therefore more sensitive to noise. Although this polygon does not have large empty areas, it has too many cavities, which result in too many edges and sharp angles. If this polygon is used as the model for the cluster, having so many edges will make the model less efficient in terms of storage and processing costs. If this polygon model is used for clustering new

samples, a new sample inside the cavities may not be assigned to this cluster although the sample is in the middle of many of the samples belonging to the cluster.

The polygon in Figure 19c, on the other hand, balances the two objectives as it does not include large empty areas and has a low degree of ruggedness. In the following, a polygon generation framework will be introduced which fits a polygon P to a set of spatial objects D minimizing the two objectives, we introduced earlier; namely, generating smooth polygons that have *a low emptiness with respect to D* and a low complexity. Additionally, we require that all objects in D are inside the polygon P. More formally, we define the problem of fitting a polygon P to a set of spatial objects as follows:

Let D be a set of spatial objects in the cluster. Our goal is to find a polygon P that minimizes the following fitness function:

$$\phi(P,D) = Emptiness(P,D) + C * Complexity(P) \qquad (9)$$

subject to the following constraint:

$$\forall o \in D: inside(o,P)$$

where C is a parameter which assesses the relative importance of polygon complexity with respect to polygon emptiness; e.g., if we assign a large value of C, smooth polygons will be preferred. *Emptiness(P,D)* is a quantitative emptiness measure that assesses the degree to which P contains empty regions with respect to D. *Complexity(P)* measures the complexity of polygon P.

In the following subsection, we first introduce a novel polygon-emptiness measure. We then describe a polygon complexity measure that has been defined by some other

work [Brinkhoff 1995], which will be reused in our work; finally, we introduce a method that fit polygons P to D, solving the optimization problem described in Eq. (9).

5.5.5 Measuring the Emptiness of a Polygon with respect to a Dataset

We have a surface, and we like to measure the emptiness in a surface with respect to spatial objects embedded into it. We call a subspace of the surface empty, if the density of the objects that are inside the subspace is low (typically, below a user-defined threshold).

One approach to measure emptiness is to associate a density function with the area covered by the polygon. However, in this work we use a different approach, which is based on Delaunay triangulations.

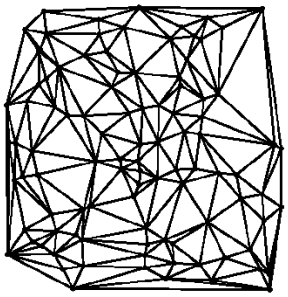


Figure 20. Delaunay triangulation of a point set

In general, as can be seen in Figure 20, areas with very low density can be identified by large triangles in the Delaunay triangulation; that is, triangles whose area is above a certain size θ. For example, if we use the average triangle size in DT(D) as the threshold, the area of the large triangle on the upper right would be identified as an empty area. We introduce an emptiness measure, which assesses the emptiness of a polygon P with respect to a point cloud D. Let:

P be a polygon whose emptiness has to be assessed,

D a set of points in 2D that P is supposed to model,

DT(D) the set of triangles of the Delaunay triangulation of D,

θ be the triangle area threshold,

$P_{CONV}=(\cup_{t\in DT(D)}\ t)$ be the outer polygon of the DT(D); P_{CONV} acts as the surface into which the objects of D are embedded and it is also the convex hull of D.

Our definition of emptiness of a polygon P with respect to a point cloud D is as follows:

$$\text{Emptiness}(P,D) := (\Sigma_{t\in DT(D)\wedge area(t)>\theta\wedge inside(t,P)}\ area(t)-\theta))/area(P_{CONV}) \qquad (10)$$

When assessing emptiness of P with respect to D, we go through the triangles inside P, and add the differences between θ and the area they cover; but, only if the size of their area is above θ, and divide this sum by the area of the convex hull of D. It should be noted that p_{CONV} is not the area P covers, but a usually larger polygon which is the union of all triangles of Delaunay triangulation which serves as the surface into which the objects of D are embedded in. It should be noted that when measuring emptiness, triangles that are not part of P trivially do not contribute to emptiness.

5.5.6 Measuring the Complexity of a Polygon

We assess the complexity of polygons using the polygon complexity measure which was introduced by Brinkhoff et al. [1995]; it defines the complexity of a polygon P as follows:

$$Complexity(P) := 0.8 * ampl(P) * freq(P) + 0.2 * conv(P) \qquad (11)$$

where $ampl(P)$ is amplitude of vibration defined as:

$$ampl(P) := 1 - (\text{boundary}(\text{convexhull}(P)) / \text{boundary}(P))$$

and *freq*(P) is the frequency of vibration of a polygon p:

$$freq(P) := 16 * (notches_{norm}(P)-0.5)^4 - 8 * (notches_{norm}(P)-0.5)^2 + 1$$

where

$$notches_{norm}(P) := notches(P) / (vertices(P)-3)$$

and a notch is defined as a vertex with an interior angle greater than 180 degrees. Lastly, convexity of a polygon P is defined as:

$$conv(P) := 1 - (area(P) / area(convexhull(P)))$$

According to this definition, polygons with too many notches, having significantly smaller areas and larger perimeters compared to their convex hulls are considered complex polygons. Most importantly, it is a suitable measure to assess the ruggedness of a polygon model generated.

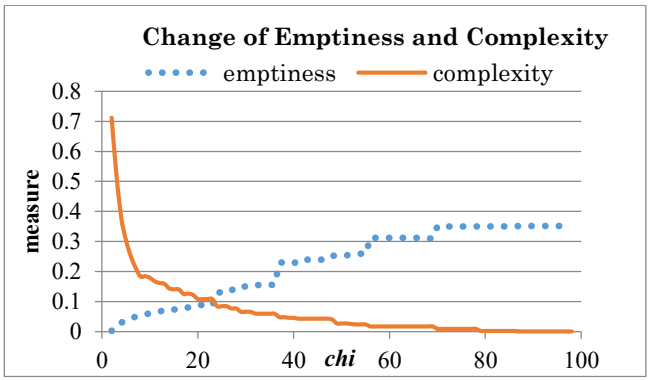


Figure 21. Change of emptiness and complexity measures of polygons created for the same cluster using different *chi* parameters

Figure 21 depicts the change of emptiness and complexity of the polygons generated using the Characteristic shapes algorithm for the cluster in Figure 19 using different *chi* parameters. Emptiness was computed by setting triangle area threshold θ to 1.25 times the average triangle area in DT(D). As seen in the figure, emptiness and

complexity of the polygons are inversely proportional. Larger chi parameters create polygons which are emptier but less complex and vice versa. We employ the fitness function defined in (9) to find a balance between the two measures.

5.5.7 Generating Polygons

We use Characteristic shapes to generate polygons in conjunction with the proposed fitness function as this algorithm produces simple polygon models. The algorithm itself has a normalized parameter *chi* which has to be set to an integer value between 1 and 100. The algorithm firstly creates uses Delaunay triangulation of the cluster and starts with the convex hull as the initial polygon (when *chi*=100 convex hull is returned as the result). It then reduces it to a non-convex hull by replacing the longest outside edges of the polygon by inner edges of the Delaunay triangulation until a termination condition is met, which is determined by *chi* parameter. Thus, smaller parameter settings result more complex polygons, and larger parameters generate larger, smoother and less complex polygons.

In order to find the value of *chi* which minimizes the employed fitness function we exhaustively test all 100 *chi* values, and return the most fit polygon as the result. Runtime complexity of the Characteristic shapes algorithm is O($n$log$n$). Thus, our methodology is guaranteed to generate the fittest polygon in O($n$log$n$) as we exhaustively test 100 parameter values.

5.5.8 Summary

In this section, we introduced two alternative methods for creating polygon models for spatial clusters. The main contributions of the work in this section include:

(1) We presented a novel methodology for creating polygon models using the Voronoi diagram when the whole dataset is available.

(2) We presented another methodology for creating polygon models for spatial clusters. A novel quantitative polygon fitness function is introduced to guide the generation of polygons from point sets, alleviating the parameter selection problem when using existing polygon generation methods. The proposed methodology uses Characteristic shapes [Duckham et al. 2008] to generate polygon models; however, we wish to emphasize that other polygon generation methods, such as the Concave Hull algorithm [Moreira and Santos 2007] and Alpha shapes [Edelsbrunner et al. 1983] can be used in conjunction with the proposed fitness function.

(3) A novel emptiness measure is introduced that quantifies the presence of empty areas in a polygon.

CHAPTER 6

EXPERIMENTAL EVALUATION

In this section, we will evaluate our methodology using three case studies. In the first case study, we find high correlation hotspots of ozone and PM 2.5 concentrations in a gridded air pollution dataset. In the second case study, we find hotspots in a crime dataset where majority of the crimes belong to a single type using the purity interestingness function. The first two experiments will be used to display the inner workings of our methodology, and evaluate its effectiveness. In the last experiment, we compare our methodology with SatScan in a case study in which we find hotspots taxi pick-up locations where the total money made per minute by taxi drivers is higher than a specified amount. In the experiments, we will evaluate the obtained hotspots based on their interestingness values and sizes. Moreover, we report wall clock runtimes of the phases of our methodology for the first and second experiment. All experiments were conducted using a MacBook Pro with an Intel i7 processor with 4 physical cores and 16 GB of RAM.

For all experiments, we use the following reward function for evaluating the quality of a region R for the hotspot growing and redundant hotspot removal phases:

$$\varphi(R) = interestingness(R) \times \text{size}(R)^{\beta} \quad (12)$$

where $\beta>1$ is a parameter determining the degree preference for larger regions. We set β to 1.01 for the experiments, giving preference to relatively smaller hotspots with high interestingness over larger hotspots with low interestingness, while preferring larger hotspot if two hotspots have the same interestingness.

6.1 Finding Correlation Hotspots in a Gridded Air Pollution Dataset

In this subsection, we present a case study in which we use a four-dimensional spatial-temporal air pollution dataset. Using a nationwide network of monitoring sites, observations for a range of climatic variables and air pollution data is stored in gridded data sets by Environmental Protection Agency (EPA), and huge amounts of gridded data are generated every day. Just the air pollution data [EPA 2016] for Houston Metropolitan area requires 1.8 GB of space for each day. This dataset contains four dimensions (longitude, latitude, altitude, and time as measurements are taken every hour) including 84 columns, 66 rows, 27 layers and 24 hours for each day. This corresponds to 3.6 million grid cells for a day and 1.3 billion grid cells for a year; moreover, each grid cell contains 132 air pollutant densities as attributes, and these observations are typically extended by adding meteorological and other types of observations for a particular analysis task, such as humidity, temperature, wind speed and solar radiation.

The goal in this case study is to find regions with high correlation of ozone and PM2.5 concentrations in the air pollution dataset. PM2.5 is abbreviation for Particulate Matter 2.5, which refers to tiny particles or droplets in the air that are 2.5 microns or less in width. PM2.5 is an air pollutant that causes health concerns when levels in air are high. We use the correlation interestingness function defined in (2) in Section 4.2 and set the interestingness threshold θ to 0.75 for this experiment to find areas with very high correlation. We use the air pollution data for 24 hours on September 1, 2013 for Houston metropolitan area grid cells and set the seed size to

3x3x3x3 grid cells. We will show results for all phases of the methodology in this experiment.

Phase 1. Identifying Neighboring Objects: We do not create a neighborhood graph for a gridded dataset, as the neighborhood relation is trivial. We used the neighborhood definition given in (6) in Section 5.1 for gridded datasets.

Phase 2. Finding Seed regions: In the whole dataset, correlation coefficient between ozone and PM2.5 concentrations is -0.47. We started with seed candidate thresholds as low as 0.75 to potentially grow smaller regions where |correlation(seed)|> 0.75, however, there were too many such regions; therefore, we set the seed candidate threshold to 0.95 to grow only the best seeds and found 235 seed regions. We merged neighboring seed regions using μ=0.96 as the merge threshold for the merge rule; from the initial 235 seeds we obtained 108 seeds by merging neighboring ones. Merging hotspot seeds took 0.16 seconds in total.

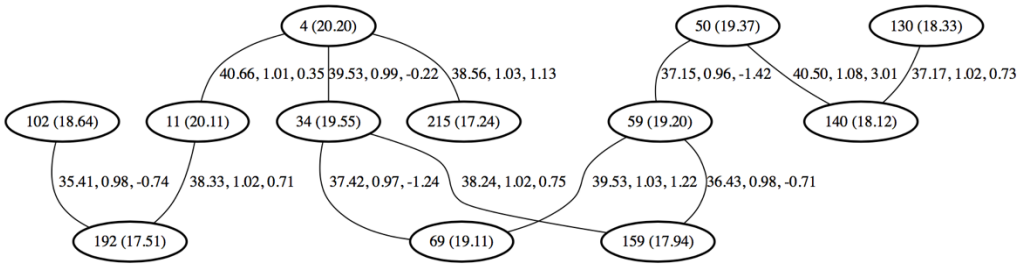


Figure 22. A part of the seed neighborhood graph for gridded datasets

Figure 22 shows a part of the seed neighborhood graph used for merging neighboring seeds. Each vertex corresponds to a seed region. First number in each vertex is the seed number and the second number in parenthesis is the reward of the

seed. There is an edge between seeds if merging the connected seeds result in a region with a reward larger than $\mu * (R_1 + R_2)$, where R_1 and R_2 are seed rewards.

For example, in the neighborhood graph, merging seeds 4 and 215 generate a new seed with a reward of 38.56 when merged, which is a 1.12 improvement (shown as 1.13 in the graph due to rounding error) obtaining a reward that is 1.03 times larger than the total reward (20.20 + 17.24= 37.54) before the merge.

Seeds are merged giving preference to the merge candidates with the highest reward gain. In this graph, seeds 4 and 215 will be merged first. Then, the edges are computed for the newly created seed region and the same process is repeated until there are no more merge candidates left.

Phase 3. Growing seed regions: After growing these 108 seed regions, we observed that many of them grew to the same boundaries. For example, 35 seed regions grew to the same hotspot which lies between x:[0,25] y:[0,18] z:[5,11] t:[0,14]. The largest hotspot had 59280 grid cells; the smallest had 1925 grid cells. Maximum growing time was 0.825 seconds for the hotspot of size 51870, and the minimum growing time was 0.018 seconds for the smallest hotspot of size 1925. The total time of growing all 108 hotspots in parallel was 23 seconds. We also turned off parallel processing and reran the experiment, which increased the total time to grow all hotspots to 73 seconds. Thus, the parallel speedup using a computer with four processors was 73/23 = 3.17.

Phase 4. Post-processing phase: We created the overlap graph for the 108 hotspots obtained using 0.66 as the overlap threshold; and the obtained graph had 1994 edges connecting the 108 vertices. Creating the graph requires finding the number of shared objects between all pairs of hotspots; since there were many hotspot with tens of

thousands of objects, the graph creation took 67 seconds. As shown in Figure 23, there are three large connected components in the graph where there is a very high number of overlaps between hotspots. We applied the graph simplification algorithm to the overlap graph and 83% of the vertices and 98.5% of edges were eliminated leaving only 18 vertices and 28 edges in the graph.

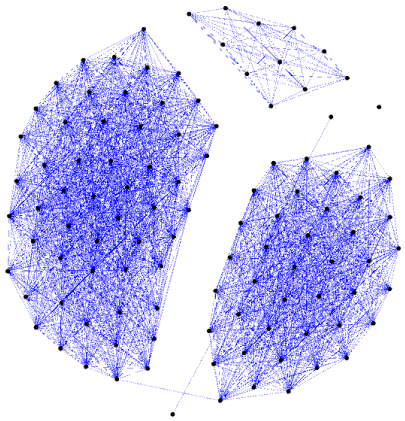


Figure 23. Overlap graph before simplification

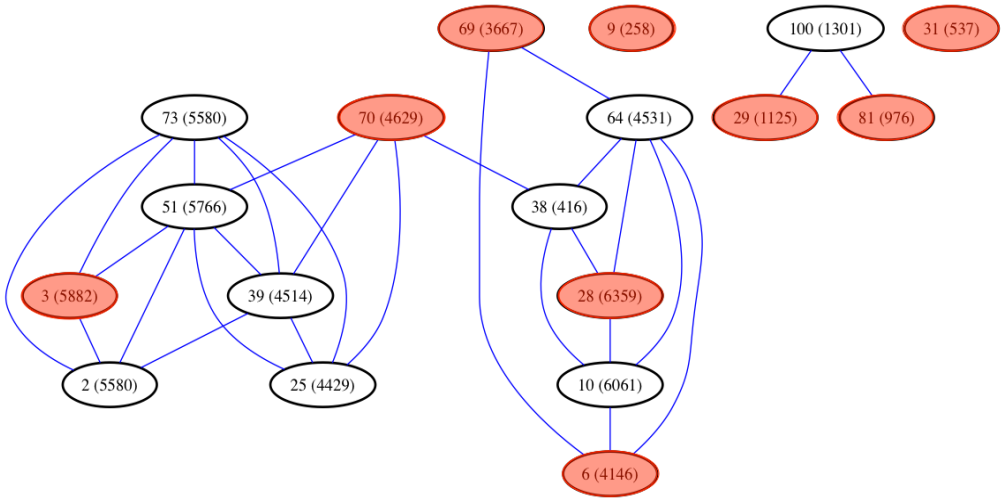


Figure 24. Simplified overlap graph. Hotspot ids and rewards (in parenthesis) are shown. Non-adjacent vertices maximizing the total reward are colored in red

Figure 24 shows the overlap graph of the remaining 18 hotspots. Running the maximum weight clique algorithm on the complement of this graph, we obtained 8 non-adjacent vertices maximizing the total reward value which are colored red in the overlap graph (hotspots numbered 3, 6, 9, 28, 29, 31, 70, 81). Running the maximum weight clique algorithm on the complement of this graph took only 10 milliseconds. Running the same algorithm on the complement of the original graph before simplification (Figure 23) with 108 vertices and 1994 edges took 1.2 seconds.

We found by performing other experiments that maximum weight clique algorithm can cope well with graphs up to a few hundreds of vertices and thousands of edges. However, it cannot cope well with graphs with more than 400 vertices and tens of thousands of edges (depending on the graph structure). In order to show the effectiveness of the graph simplification algorithm, we conducted another experiment in which we changed the seed threshold to 0.85 and skipped seed merging step, obtaining 563 hotspot seeds. After growing these hotspots, we obtained an overlap graph with 563 vertices and 41120 edges. Running the maximum weight clique algorithm on the complement graph without simplifying this graph took 30 hours. Next, we simplified this graph using our graph simplification algorithm, we obtained a graph with 57 vertices, and 238 edges—the graph simplification algorithm took only 0.52 seconds on this graph. Running the maximum weight clique algorithm on the complement of simplified graph took under a second, and the results were exactly same. This is, we believe, a spectacular improvement, and shows the effectiveness of the graph simplification algorithm.

Phase 5. *Finding the Scope of Hotspots:* The scope of hotspots created by the dimensional growing algorithm is the rectangular boundary of the hotspot region.

Table 1 shows the properties of final set of 8 hotspots. Hotspot numbers, correlation of ozone and PM2.5 in the hotspot seeds, hotspot rewards, correlation of ozone and PM2.5 levels in the hotspots, total time to grow the hotspots and the scope of hotspots are listed.

Table 1. Properties of correlation hotspots

HS#	Seed Correl.	Correlation	Reward	Total Cells	Growing time (ms)	Scope of Hotspot
3	0.96	-0.89	5882	38038	493	x:[0,25] y:[0,18] z:[0,6] t:[3,13]
6	0.97	0.91	4146	23104	274	x:[0,7] y:[0,18] z:[4,11] t:[0,18]
9	0.95	0.87	258	1925	18	x:[0,4] y:[12,18] z:[1,11] t:[15,19]
28	0.97	0.88	6359	44460	696	x:[0,25] y:[0,18] z:[6,11] t:[0,14]
29	0.97	0.86	1125	9120	183	x:[16,25] y:[0,18] z:[0,11] t:[20,23]
31	0.96	0.8	537	9072	81	x:[0,23] y:[13,18] z:[3,11] t:[17,23]
70	-0.95	-0.89	4629	29260	377	x:[4,25] y:[0,18] z:[0,9] t:[9,15]
81	0.97	0.82	976	12750	192	x:[9,25]vy:[0,14] z:[2,11] t:[19,23]

The largest hotspot (#28) contains 44460 grid cells which corresponds to a very large area including all longitudes and latitudes between layers 6 to 11, from 12am to 2pm on the selected date. Ozone levels are highly correlated with PM2.5 levels at this time and a domain expert can do further analysis to find out the reasons. On the other hand, hotspot #3 containing 38038 grid cells indicates a region with highly negative

correlation (-0.89) between ozone and PM2.5 levels. This hotspot is located at the same latitude and longitudes but in layers 0 to 6 at about the same timeframe (3am to 1pm).

Figure 25 shows the graph of growing time of hotspots and hotspot sizes. The runtime complexity of dimensional hotspot growing algorithm is very close to O(n).

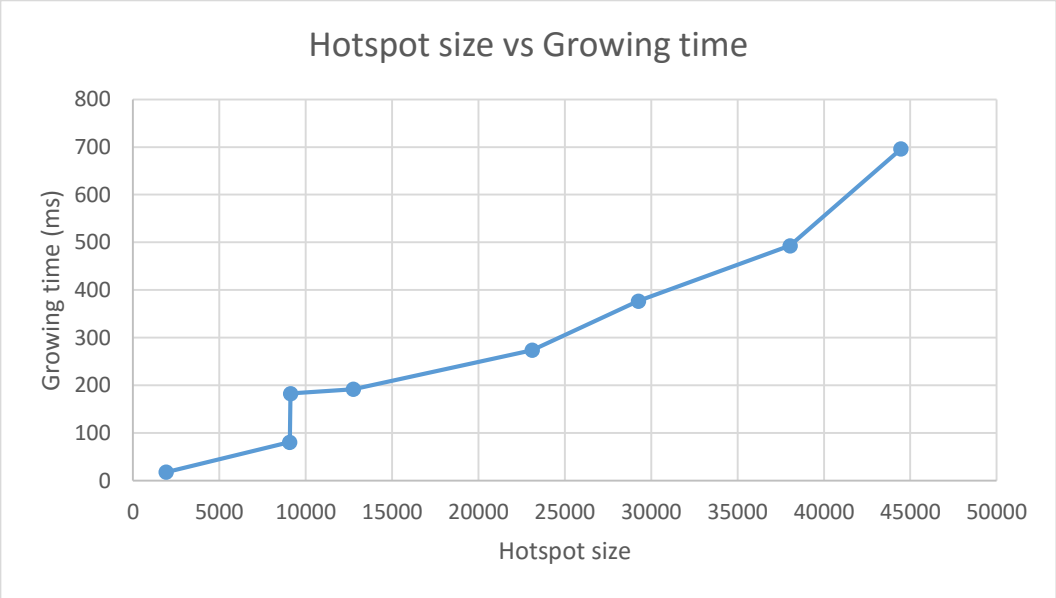


Figure 25. Dimensional growing times by hotspot size

Figure 26 illustrates hotspots 3 (blue), 6 (gray) and 9 (red). Time dimension was not taken into account in this figure, thus all hotspots seem to overlap more than they do. For example, hotspot 3 occurs from 3am to 2pm, whereas hotspot 9 occurs from 3pm to 8pm, thus they do not overlap. Dimensional growing algorithm creates rectangular hotspots as in Figure 26, which are easy to comprehend. On the other hand, runtime complexity of the dimensional growing algorithm is very close to linear, which makes it an attractive option with very large gridded datasets. Moreover, our framework allows growing gridded hotspots using a subset of dimensions—seed can be created in 2 or 3 dimensional spaces, and can be set to grow only in x or y dimensions

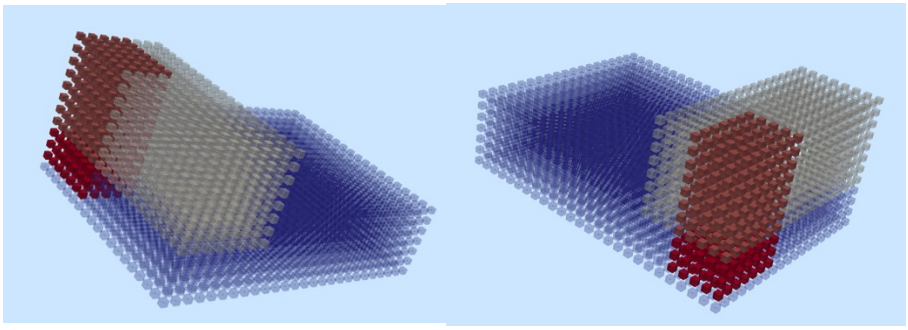


Figure 26. Three correlation hotspots from different angles (time dimension not shown)

Heap-based growing algorithm can also be used with gridded datasets. However, hotspots created using heap-based growing algorithm will have irregular shapes, and these hotspots will be larger, as adding the best neighbor in each step will allow hotspots to exclude many of the grid cells that has to be included in dimensional growing, keeping the interestingness high and allowing for more controlled growing. We refer to our preliminary work [Akdag et al. 2014] for more details about growing irregularly shaped hotspots in a gridded dataset.

6.2 Finding Purity Hotspots in a Crime Dataset

In this experiment, we use purity interestingness function (4) defined in section 4.2 to find hotspots in a crime dataset where majority of the crime events belong to a single type of crime. The goal of this experiment is to show how our methodology employs a neighborhood graph, and grows seeds identified in this graph using the heap-based growing algorithm. We use Montgomery County of Maryland [2016] Crime data from 1/1/2014 to 3/29/2016. There are 4910 crime events belonging to 27 different types of crimes including assault, illegal substance use, robbery, larceny, etc. Table 2 shows the

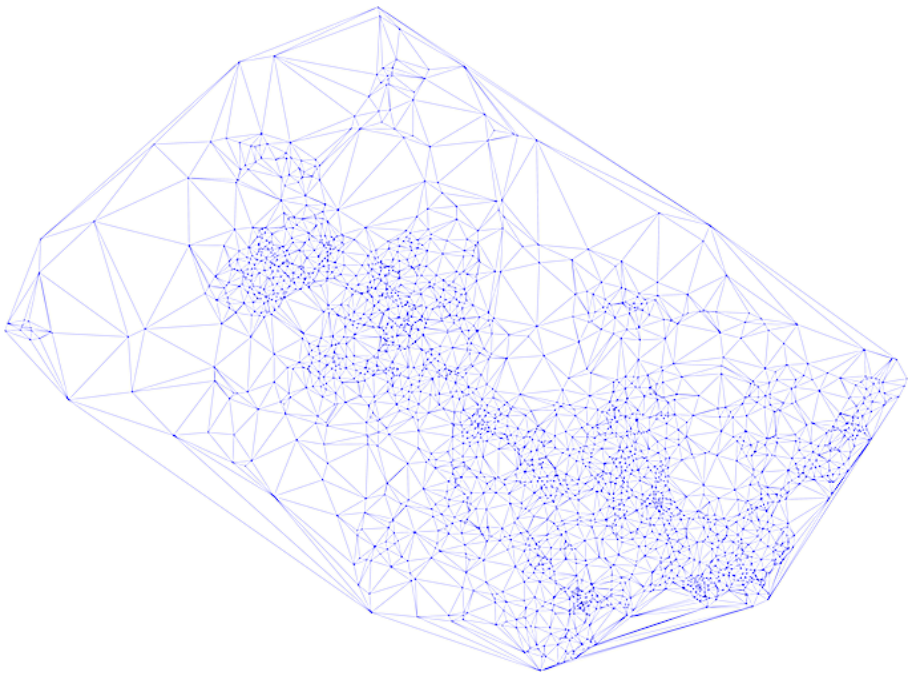
total numbers and percentages of crimes by category type. We set purity threshold to 0.66 to find hotspots where at least 2/3 of the crimes are of same type.

Table 2. Crime event counts and rates in the data set

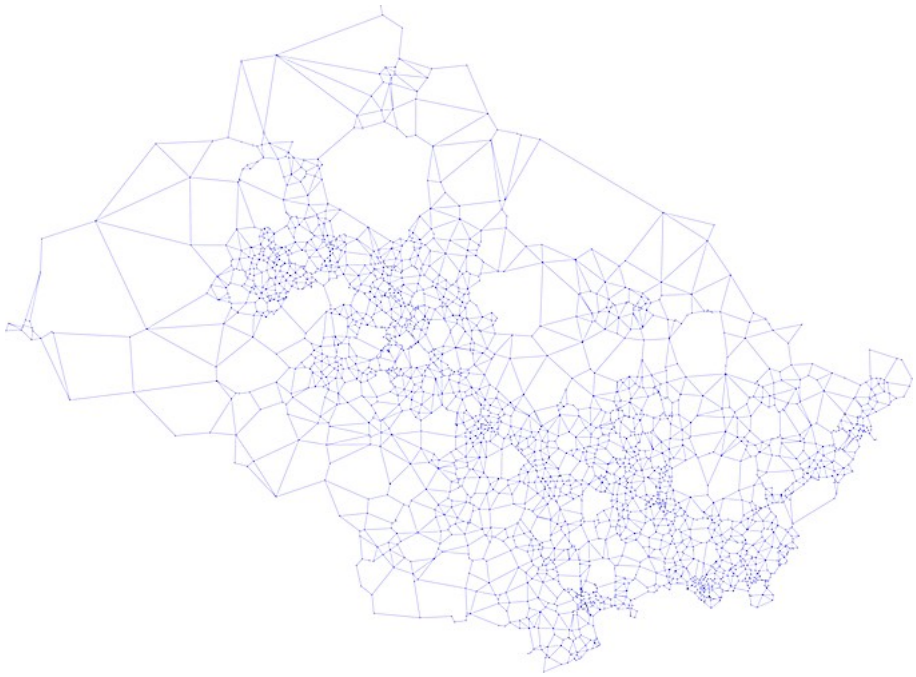
Category	**count**	**rate**
Larceny	1282	26.11%
Substance use	416	8.47%
Driving under influence	341	6.95%
Forgery	334	6.80%
Vandalism	290	5.91%
Assault	272	5.54%
Mental Transport	236	4.81%
Burglary	186	3.79%
Loss of Property	181	3.69%
Trespassing	170	3.46%
Illegal Drinking	140	2.85%
Disorderly conduct	117	2.38%
16 other categories	889	19.24%
Total	4910	100%

We will present the results for all 5 phases of our methodology in this experiment.

Phase 1. Identifying neighboring objects: Figure 27 shows the Delaunay triangulation and Gabriel graph for the dataset. As shown in the figure, Delaunay triangulation contains connections between many distant points, especially around the edges of the point set. However, those connections were removed in the Gabriel graph. We use the Gabriel graph for defining the neighborhood of points in the dataset.



a) Delaunay triangulation



b) Gabriel Graph

Figure 27. Delaunay triangulation and Gabriel graph for the Crime data set

Phase 2. Finding seed regions: A seed candidate region is created around each point which consists of the point and its first degree neighbors. Out of 4910 seed candidates, 217 of them had an interestingness larger than the seed threshold which we set to 0.66. 99 of the neighboring seeds were merged based on the reward increase and as a result 128 hotspot seeds were obtained.

Phase 3. Growing seed regions: When growing a hotspot, we use a heap data structure to store the neighbors and we assign each neighbor a priority value based on how they fit to the region, which determines their location in the heap. We used the following fitness function to evaluate the fitness of a neighbor n_i of hotspot H:

$$fitness\ (n_i) = (i(H \cup n_i) - i(H)) \qquad (13)$$

where $i(H \cup n_i)$ represents the new interestingness when n_i is added to the hotspot and $|H|$ is the hotspot size. The neighbors which increase the interestingness value most is given priority, and added to the region first.

When 128 hotspot seeds were grown, the largest four hotspots all had 36 events. Many hotspots grew to the same region. Thus, we eliminated such highly overlapping redundant hotspots in the next phase.

Phase 4. Post-processing: We simplified the overlap graph using 0.66 as the overlap threshold. The resulting graph had only 24 vertices and 8 edges. 104 hotspots were eliminated in this phase. Using the maximum weight clique algorithm on the complement of this graph, we obtained 19 non-adjacent vertices maximizing the total weight. Out of the 19 hotspots, 17 of them belongs to larceny crime type, which is the most common crime type. The remaining two crimes belonged to "driving under influence" crime type. The average size of a hotspot was 12.

Phase 5. Creating hotspot scopes: We created polygonal boundaries for hotspots by merging the Voronoi polygons for the points in each hotspot. Figure 28 shows the polygons that represent two hotspots where majority of the crimes belong to "driving under influence" crime category and Figure 29 shows the Voronoi regions which were merged to create the yellow polygon in this figure. Figure 30 shows these events in the dataset; as shown in the figure, yellow colored points have the majority of events in those areas, which proves that the hotspots were correctly identified.

Figure 28. Polygon created for "driving under influence" crime purity hotspots, colored in red and yellow

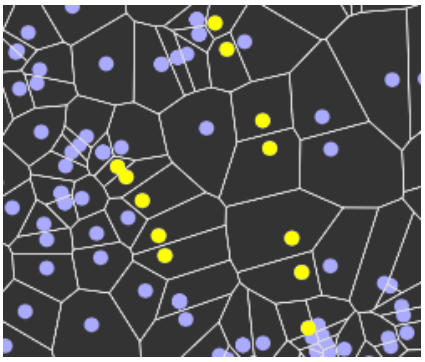


Figure 29. Voronoi regions for the yellow polygon in Figure 28

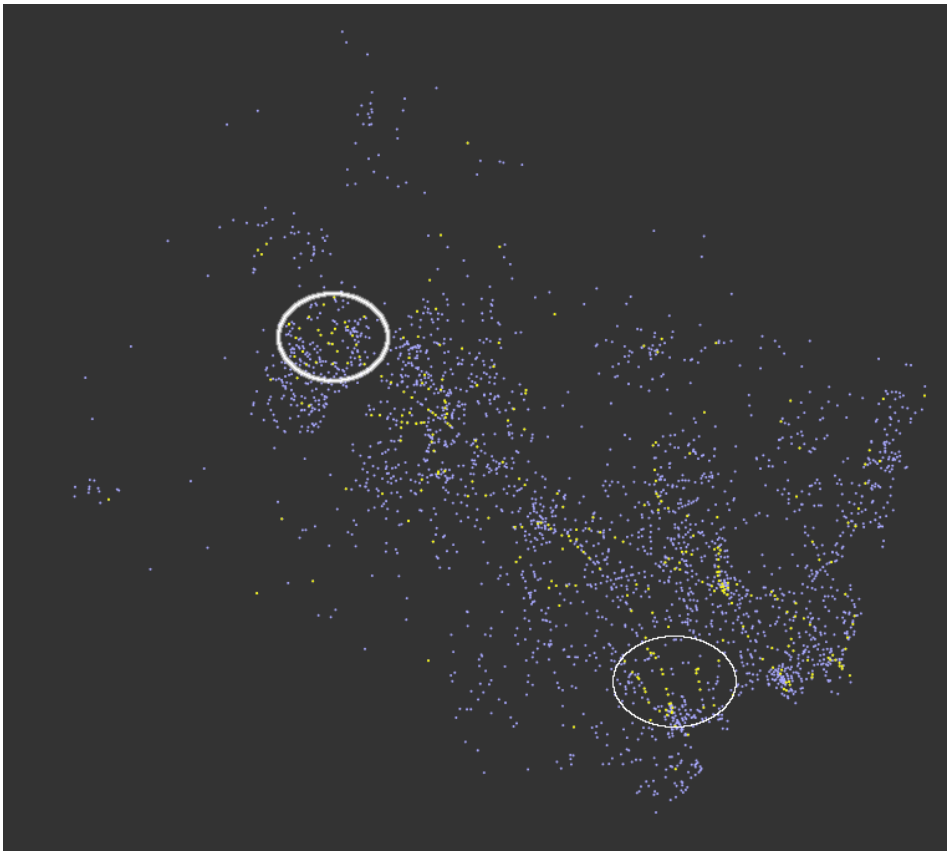


Figure 30. “Driving under influence” crime locations colored in yellow color. Circles highlight areas where majority of the crimes are of this type

Summary: In this experiment, we used a graph-based hotspot discovery approach to find purity hotspots in a point-based crime dataset. The results show that our framework was able to detect the hotspots accurately. We created hotspot scopes using a method that is based on Voronoi diagram and convex hull of the dataset. To the best of our knowledge, this is the only hotspot discovery algorithm that uses Gabriel graphs to define neighborhood relations, and grows hotspots using the seeds identified in this graph. Moreover, the proposed method for creating hotspot scopes is the only approach that creates polygonal hotspot scopes using Voronoi diagram and convex hull.

6.3 New York Taxicab Dataset: Comparison to SatScan

In this experiment, we will compare our framework with the state-of-the-art hotspot discovery tool SatScan, and show the differences. We will use an interestingness function that calculates interestingness of a hotspot based on a predefined attribute. As discussed before, SatScan can also be used for finding hotspots with interestingness functions that are defined using a single attribute of the objects in the dataset. The hotspots created by SatScan and our framework will be evaluated based on hotspot size and interestingness. For this experiment, we will skip details for each phase and just show and compare the final results.

We will use New York City taxicab dataset [NYC Taxi and Limousine Commission 2016] which contains trip records including fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types collected from green taxi trips from 1/1/2016 to 1/22/2016. In this experiment, we will find hotspot locations in which average money made per minute is more than 1.5 dollars (which is 1.01 dollars/min in the input data). We use the following formula to calculate the average money made by minute for a trip:

$$Rate(o) = \frac{total\ fare\ including\ tips - total\ cost\ of\ trip}{duration\ in\ minutes}$$

where total cost is estimated as gas cost (gas price * total distance in miles / mpg) plus toll fees; and gas price was set to $2 per gallon and mpg was set to 20 miles per gallon. We use the Rate value as the attribute value for each object in SatScan, and configure SatScan to find hotspots with high rates. Our framework calculates the interestingness of a hotspot as the average rate in the hotspot:

$$i_p(\mathrm{H}) = \frac{\sum_{h \in H} h.p}{|H|}$$

where $h.p$ is the Rate of each object h in the hotspot. The fitness of a neighboring object was assigned to its Rate.

There were 1,048,574 trip records in the dataset. We randomly chose 10,000 records, eliminated instances with errors and used the resulting 9626 trip records as the input. Pickup locations of the trip records and Voronoi diagram of all points are shown in Figure 31. Figure 32 shows the distribution of money made by minute in 0.1 dollar intervals.

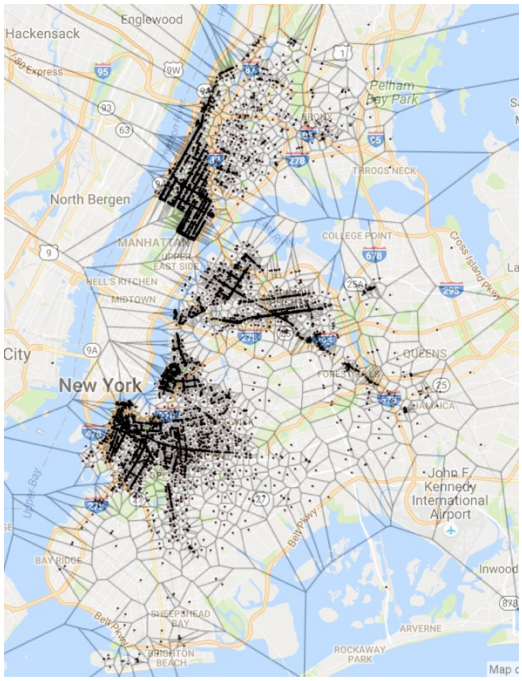


Figure 31. New York green taxicab dataset and Voronoi diagram for the dataset

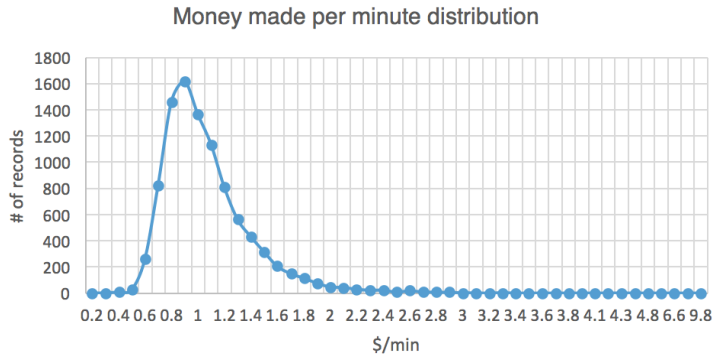


Figure 32. Distribution of dollars made by minute in the dataset

Results using our framework: Our framework detected 32 non-overlapping hotspots with at least five records in each. We will only visualize and report the five hotspots with more than 10 objects in the result. Properties for each hotspot is given Table 4 and Figure 33 shows the hotspots on the map.

Table 3. Taxicab hotspots discovered by our framework

Hotspot	Rate ($/min)	# of records
1	1.62	12
2	1.58	22
3	1.57	13
4	1.60	15
5	1.69	11

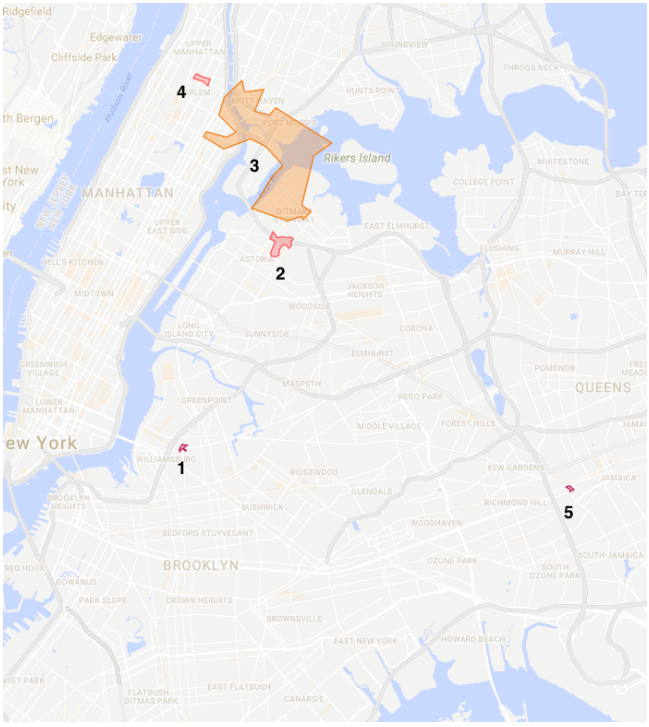


Figure 33. Taxicab interestingness hotspots detected by our framework.
Only the hotspots with more than 10 records are shown.

SatScan results: We firstly used all 9626 trip records with SatScan and obtained only two very small hotspots with 2 objects in each. SatScan is very sensitive to extreme outliers, and created those two hotspots around trip records with extremely high rates as shown in Table 4 and Figure 34:

Table 4. Properties of hotspots discovered by SatScan for taxicab dataset without removing outliers

Hotspot	Rate ($/min)	#of records	Radius (km)
1	5.48	2	0.061
2	5.39	2	0.16

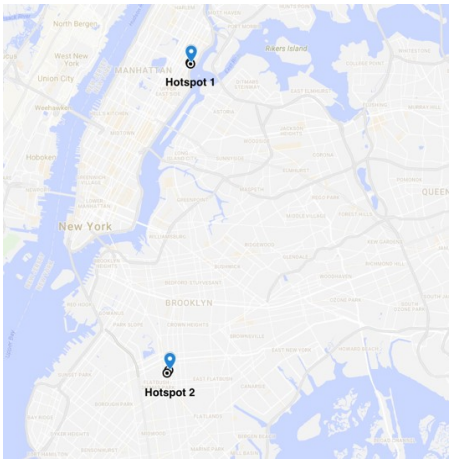


Figure 34. SatScan results for taxicab dataset without removing outliers

To obtain larger hotspots, we deleted 27 trip records, which had a rate more than 3$/minute and ran SatScan again obtaining 7 hotspots with largely varying sizes as shown in Table 5. Hotspots detected by SatScan after removing outliers and Figure 35.

Table 5. Hotspots detected by SatScan after removing outliers

Hotspot	Rate ($/min)	# of objects
1	1.31	262
2	1.28	187
3	1.18	924
4	2.09	4
5	1.98	4
6	1.88	5
7	2.28	2

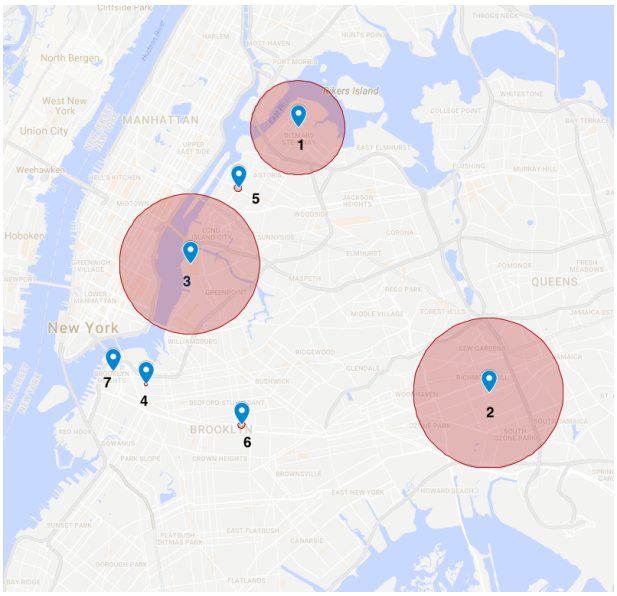


Figure 35. Hotspot locations detected by SatScan after removing outliers. Three large areas has interestingness values less than the interestingness threshold ($1.5/min); thus, they are not considered as interestingness hotpots.

Three hotspots were very large containing hundreds of locations but the rate was at most 1.31$/min in these hotspots. Four very small hotspots containing less than 10 objects were detected with rates higher than 1.5$/minute. These small hotspots were also detected by our framework but we discarded them as being too small. When we reran our framework with the new dataset with 27 trip records deleted, the results were not significantly different.

An interesting observation is that the large region with a low rate detected by SatScan on the very north of the map (hotspot 1) overlaps with 2 polygonal interestingness hotspots detected by our framework in the same area (Figure 36). This is an indication of the power of detecting irregularly shaped hotspots using an interestingness function that drives the growing process. Our framework is able to choose which areas are interesting and can grow the hotspot in that direction—e.g., in

north-west direction of the SatScan hotspot—, whereas SatScan cannot exclude areas with low interestingness in the circular hotspot.

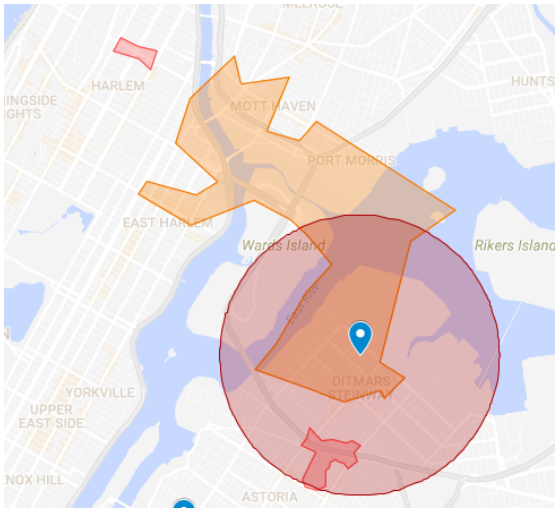


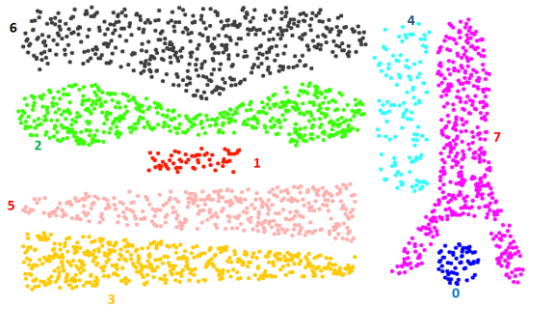
Figure 36. Hotspots detected by SatScan (circular) and our framework (polygons) in the same region

Comparison: The results show that our framework is able to detect better interestingness hotspots compared to SatScan. The hotspots detected by our framework were larger in size and area, and they had acceptable rates even in presence of outliers. SatScan is very sensitive to outliers and creates very small hotspots around records with very high values, ignoring others. Even after removing outliers, it was unable to detect large hotspots with high interestingness value; the large hotspots it detected had low interestingness and there is no way to adjust SatScan to detect hotspots with higher than a threshold rate. This is not surprising as our framework is specialized in detecting irregularly shaped hotpots maximizing a given function, whereas SatScan tries to find circular regions that have statistically very high number of events compared to the rest of the dataset.

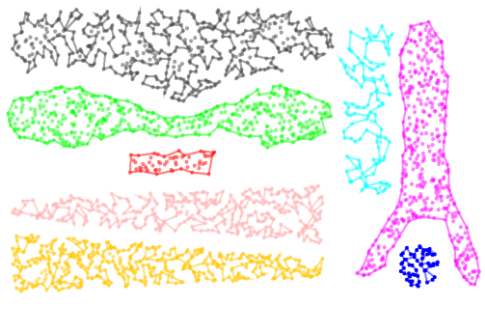
6.4 Creating Polygon Models for a Set of Clusters

In this section, we present a case study in which we evaluate the methodology proposed for creating polygon models for spatial clusters. We will present the experimental results using the fitness function ϕ defined in equation (9) in section 5.5.4. We will find polygon models for clusters in Complex8 dataset [Salvador et al. 2004].

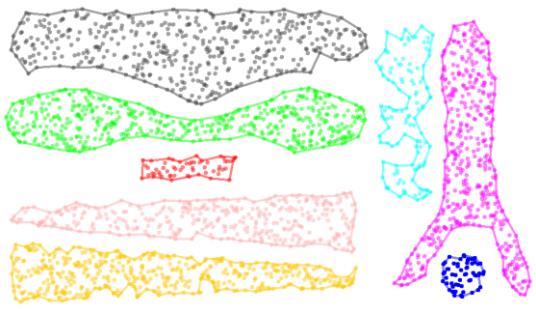
The dataset used for this experiment is depicted in Figure 37a. Figure 37b-d depict the polygons generated for each cluster using different C parameters.



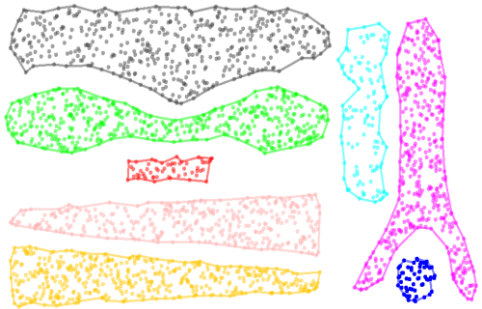
a) Complex 8 clusters



b) Polygons created for C=0.2



c) Polygons created for C=0.35



d) Polygons created for C=0.5

Figure 37. Complex 8 Dataset and polygons generated using different chi parameters

Table 6 reports the area, perimeter, emptiness, and complexity for polygons in these figures along with the optimal *chi* parameter values selected by the fitness function to create these polygons.

Table 6. Statistics for polygons in Figure 37b-d separated by comma in respective order. P0-P7 represent polygons for clusters 0-7 in the dataset and colored respectively.

	area	perimeter	emptiness	complexity	chi
P0	1088, 2030, 2030	328, 173, 173	0.077, 0.219, 0.219	0.49, 0.02, 0.02	37, 70, 70
P1	2697, 2697, 2741	287, 287, 286	0.144, 0.144, 0.148	0.052, 0.052, 0.046	34, 34, 37
P2	21492, 23107, 23107	1052, 997, 997	0.084, 0.096, 0.096	0.125, 0.072, 0.072	6, 13, 13
P3	9477, 18057, 20146	2465, 1058, 954	0.072, 0.192, 0.233	0.589, 0.118, 0.02	2, 5, 10
P4	4829, 8408, 11246	1171, 751, 561	0.057, 0.113, 0.197	0.562, 0.319, 0.089	5, 10, 16
P5	9007, 19122, 20413	2460, 968, 947	0.063, 0.19, 0.211	0.606, 0.043, 0.015	2, 8, 18
P6	17560, 34719, 35061	3019, 1018, 1003	0.044, 0.162, 0.168	0.632, 0.054, 0.04	4, 13, 14
P7	19759, 19759, 20807	1003, 1003, 984	0.042, 0.042, 0.049	0.188, 0.188, 0.17	8, 8, 14

The polygon P4 (cyan-colored) best illustrates the effect of changing the C parameter. The generated polygon for P4 in Figure 37b is very tight and rugged having a smaller area, larger perimeter, smaller emptiness and larger complexity values compared to polygons generated with larger C values. On the other hand, the generated polygon for P4 in Figure 37d is smoother; it has fewer edges and empty areas producing a larger area and emptiness value, smaller perimeter and a smaller complexity value. In general, by adjusting the C parameter, more or less complex polygons can be created depending on application requirements.

CHAPTER 7

CONCLUSION

In this dissertation, we presented a computational framework for discovering interestingness hotspots in spatial datasets. Interestingness hotspots are contiguous regions in space, which are interesting based on a domain expert's notion of interestingness which is captured by an interestingness function. Our framework uses a hotspot growing algorithm which works by growing seed regions using plugin interestingness and reward functions. To the best of our knowledge, this is the only hotspot discovery algorithm in the literature that grows seed regions using a reward function. This fact distinguishes our approach from traditional hotspot discovery algorithms. We claim that the proposed framework is capable of identifying a much broader class of hotspots, which cannot be identified by traditional distance-based clustering algorithms and spatial scan statistics. Moreover, the proposed framework is very generic and can be used with any dataset in which a neighborhood relation between spatial objects can be defined. We proposed using Gabriel graph for defining the neighborhood of objects in two-dimensional point-based datasets. We are also exploring the usage of three-dimensional Gabriel graphs for three-dimensional point-based datasets, and partitioning the neighborhood graph for very large datasets.

We use efficient data structure like heaps and hash sets to improve the efficiency of algorithms employed in various phases of the hotspot discovery process. Moreover, we grow hotspots in parallel using a shared memory parallel processing approach, which improves the total runtime of the hotspot growing phase significantly. We are

investigating using a distributed parallel programming approach for growing all hotspots in parallel. Furthermore, we also introduced an agglomerative seed merge algorithm that decreases the number of seeds grown. We plan to investigate alternative hotspot growing approaches in our future work.

We evaluated our framework in case studies using real datasets and demonstrated that our framework is able to identify the locations of hotspots correctly and efficiently. We compared our framework with the state of the art hotspot discovery tool SatScan in a case study and our framework was able to detect better hotspots which were not detected by SatScan. Our graph-based hotspot growing algorithm which is driven by a reward and interestingness function was able to detect hotspots in different shapes while retaining the interestingness of the hotspots higher than a threshold value.

We believe that polygons are quite essential in spatial analysis and we provided two methods for creating polygon models for two-dimensional spatial hotspots and clusters. First, we presented a novel method for creating polygon models using the Voronoi diagram for spatial clusters and hotspots when the whole dataset is available. Then, we introduced a methodology for creating polygon models for any spatial cluster, which employs a novel quantitative polygon fitness function to guide the generation of polygons from point sets, and alleviates the parameter selection problem when using existing polygon generation methods. Moreover, a novel polygon-emptiness measure is introduced that quantifies the presence of empty areas in a polygon. The proposed methodology can be used with other polygon generation methods, such as Concave Hull and Alpha shapes algorithms in conjunction with the proposed polygon fitness function.

We also proposed a novel and unique graph-based post-processing algorithm that finds an optimal set of hotspots which are allowed to overlap to a degree less than a threshold value. The proposed algorithm reduces removing redundant hotspots to the maximum weight clique problem. Moreover, it improves the efficiency of Östergård's maximum weight clique algorithm [2002] significantly by simplifying the overlap graph of the hotspots. Moreover, this generic algorithm can be used in conjunction with other hotspot discovery algorithms that detect overlapping hotspots. We are also investigating the usage of this algorithm with cluster-aggregation problem, which aims to take advantage of a set of different clusterings that have been found for a data set to find a consensus clustering that is better than existing clusterings.

REFERENCES

Deepak Agarwal, Andrew McGregor, Jeff M. Phillips, Suresh Venkatasubramanian, and Zhengyuan Zhu. 2006. Spatial scan statistics: Approximations and performance study. *In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 24-33.

Fatih Akdag and Christoph. F Eick. 2014. A Computational Framework for Finding Interestingness Hotspots in Large Spatio-Temporal Grids. *In Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial-2014)*. ACM, Dallas, TX, 21-29.

Harith Alani, Christopher B. Jones, and Douglas Tudhope. 2001. Voronoi-based region approximation for geographical information retrieval with gazetteers. *International Journal of Geographical Information Science*. Vol 15, No. 4, 287-306.

Paul K., Amalaman, and Christoph F. Eick. 2015. HC-edit: A Hierarchical Clustering Approach to Data Editing. *In International Symposium on Methodologies for Intelligent Systems*. Springer International Publishing, 160-170.

Jeffrey Barrell and Jon Grant. 2013. Detecting hot and cold spots in a seagrass landscape using local indicators of spatial association. *Landscape ecology*, 28(10).

Derya Birant and Alp Kut. 2007. ST-DBSCAN: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1), 208-221.

Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. 1999. The maximum clique problem. In *Handbook of combinatorial optimization* . Springer US, 1-74.

Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Alexander Braun. 1995. Measuring the Complexity of Polygonal Objects. In *Proc. of the Third ACM International Workshop on Advances in Geographical Information Systems*, 109-117.

Zechun Cao, Sujing Wang, Germain Forestier, Anne Puissant, and Christoph F. Eick. 2013. Analyzing the Composition of Cities Using Spatial Clustering. *In Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*. ACM, Chicago, 2013.

A. Ray Chaudhuri, Bidyut Baran Chaudhuri, and Swapan K. Parui. 1997. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *Computer Vision and Image Understanding*. Vol. 68, 57–275.

Bernard Chazelle. 1993. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*. Vol. 10, No. 4, 377-409.

Chun-Sheng Chen, Nauful Shaikh, Panitee Charoenrattanaruk, Christoph F. Eick, Nouhad J. Rizk, and Edgar Gabriel. 2011. Design and Evaluation of a Parallel Execution Framework for the CLEVER Clustering Algorithm. In Proceedings of Parallel Computing Conference (PARCO-2011). Ghent, Belgium, 73-80.

Chun-Sheng Chen, Vadeerat Rinsurongkawong, Christoph F. Eick, and Michael D. Twa. 2009. Change Analysis in Spatial Data by Combining Contouring Algorithms with Supervised Density Functions. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, Berlin Heidelberg, 907-914.

Jiyeon Choo, Rachsuda Jiamthapthaksin, Chun-sheng Chen, Oner Ulvi Celepcikay, Christian Giusti, and Christoph F. Eick. 2007. MOSAIC: A Proximity Graph Approach to Agglomerative Clustering. *In Proceedings of 9th International Conference on Data Warehousing and Knowledge Discovery*. Springer Berlin Heidelberg, 231-240.

Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998. DeWall: A fast divide and conquer Delaunay triangulation algorithm in E d." *Computer-Aided Design* Vol. 30, No. 5, 333-341.

Thomas H. Cormen. 2009. Introduction to Algorithms. MIT Press and McGraw-Hill, 2009.

Montgomery County of Maryland. 2016. Crime data. Data catalog. Retrieved November 2, 2016 from http://catalog.data.gov/dataset/crime

Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. 2008. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. Pattern Recognition vol. 41, 3224-3236. (2008)

Luiz Duczmal and Renato Assuncao. 2004. A simulated annealing strategy for the detection of arbitrary shaped spatial clusters. Computational Statistics and Data Analysis, 45: 269–286.

John Eck, Spencer Chainey, James Cameron, and R. Wilson. 2005. Mapping crime: Understanding hotspots. *Academic Press*. 1-71.

Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. 1983. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*. Vol. 29, 551–559

Emre Eftelioglu,Shashi Shekhar, Dev Oliver, Xun Zhou, Michael R. Evans, Yiqun Xie, James M. Kang, Renee Laubscher, and Christopher Farah. 2014. Ring-Shaped Hotspot Detection: A Summary of Results. *In Proceedings of IEEE International Conference on Data Mining (ICDM 2014).* IEEE, 815–820.

Emre Eftelioglu, Shashi Shekhar, Xun Tang. 2016. Crime Hotspot Detection: A Computational Perspective. *In Data Mining Trends and Applications in Criminal Science and Investigations*. IGI Global, 82-111.

EPA. 2016. Air Data: Air Quality Data Collected at Outdoor Monitors Across the US. Retrieved November 2, 2016 from https://www.epa.gov/outdoor-air-quality-data/

Levent Ertöz, Michael Steinbach, and Vipin Kumar. 2003. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. *In Proceedings of 2nd SIAM International Conference on Data Mining*. 47-58.

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 Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*. Vol. 96, 226-231.

David W. Matula, Robert R. Sokal. 1980. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis*, Vol. 12(3), 205-222

Michael R. Gary, and David S. Johnson. 1979. A Guide to the Theory of NP-Completeness. *WH Freeman*, New York.

K. Ruben Gabriel, and Robert R. Sokal. 1969. A New Statistical Approach to Geographic Variation Analysis. *Systematic Biology*. Vol. 18(3), 259-278.

Rebecca L. Hale, Nancy B. Grimm, Charles J. Vörösmarty, and Balazs Fekete. 2015. Nitrogen and phosphorus fluxes from watersheds of the northeast us from 1930 to 2000: Role of anthropogenic nutrient inputs, infrastructure, and runoff. *Global Biogeochemical Cycles*, 29(3):341-356.

Vijay S. Iyengar. 2004. On detecting space-time clusters. In Proceedings of the 10th ACM SIGMOD International Conference on Knowledge Discovery and Data Mining. Seattle, Washington, 587-592.

Jerzy W. Jaromczyk, and Godfried T. Toussaint. 1992. Relative neighborhood graphs and their relatives. *In Proceedings of the IEEE* 80(9). 1502-1517.

Leonard Kaufman, and Peter Rousseeuw. 1987. Clustering by means of medoids. *Statistical Data Analysis Based on the L1 Norm and Related Methods*. North-Holland, Amsterdam, 405–416.

Martin Kulldorff. 1997. A spatial scan statistic. *Communications in statistics: Theory and Methods*. Vol. 26, 1481–1496.

Martin Kulldorff, L. Huang, L. Pickle, and L. Duczmal. 2006. An elliptic spatial scan statistic. *Statistics in Medicine* , 25(22), 3929–3943. DOI:10.1002/sim.2490

Martin Kulldorff, Lan Huang, Linda Pickle, and Luiz Duczmal. 2006. An elliptic spatial scan statistic. *Statistics in medicine* 25, No. 22, 3929-3943.DOI:10.1002/sim.2490

Ruth Miller, ChunSheng Chen, Christoph F. Eick, and Abraham Bagherjeiran. 2011. A framework for spatial feature selection and scoping and its application to geo-targeting. *In IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM).* IEEE, 26-31.

Adriano Moreira and Maribel Yasmina Santos. 2007. Concave hull: a k-nearest neighbours approach for the computation of the region occupied by a set of points. In *International Conference on Computer Graphics Theory and Applications GRAPP (2007).*

Daniel B. Neill, and Andrew W. Moore. 2004. Rapid detection of significant spatial clusters. *In Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '04)* ACM, New York, NY, 256-265. DOI: http://doi.org/10.1145/1014052.1014082

Daniel B. Neill. 2009. An empirical comparison of spatial scan statistics for outbreak detection. *International journal of health geographics.* Vol. 8, No. 1.

NYC Taxi and Limusine Commission. 2016. TLC Trip Record Data. The City of New York. Retrieved November 2, 2016 from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

Dev Oliver, Shashi Shekhar, Xun Zhou, Emre Eftelioglu, Michael R. Evans, Qiaodi Zhuang, James M. Kang, Renee Laubscher, and Christopher Farah. 2014. Significant Route Discovery: A Summary of Results. *In Proc. of GIScicence*. Springer, 284–300.

Patric R.J. Östergård. 2002. A Fast Algorithm for the Maximum Clique Problem. *Discrete Applied Mathematics*. No 1, 197-207.

GP Patil and C. Taillie. 2004. Upper level set scan statistic for detecting arbitrarily shaped hotspots. *Envir Ecol Stat*, 11:183–197.

Philippe Pébay. 2008. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. *Sandia Report SAND2008-6212, Sandia National Laboratories, 94*.

Stan Salvador, and Philip Chan. 2004. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. *In Tools with Artificial Intelligence. ICTAI 2004. 16th IEEE International Conference*. IEEE. 576-584.

Shashi Shekhar and Sanjay Chawla. 2003. Spatial databases: a tour. *Upper Saddle River*, Prentice hall, NJ.

Shashi Shekhar, Michael R. Evans, James M. Kang, and Pradeep Mohan. 2011. Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. Vol 1, No. 3 (2011),193-214. DOI: 10.1002/widm.25.

Toshiro Tango and Kunihiko Takahashi. 2005. A flexibly shaped spatial scan statistic for detecting clusters. *International Journal of Health Geographics*.

Sujing Wang, Chun-Sheng Chen, Vadeerat Rinsurongkawong, Fatih Akdag, and Christoph F. Eick. 2010. A Polygon-based Methodology for Mining Related Spatial Datasets. *In Proceedings of ACM SIGSPATIAL International Workshop on Data Mining for Geoinformatics*. (DMG), San Jose.

B. P. Welford. 1962. Note on a method for calculating corrected sums of squares and products. In *Technometrics,* 1962, 419-420.

Takahiro Yabe, Kota Tsubouchi, Akihito Sudo and Yoshihide Sekimoto. 2016. A Framework for Evacuation Hotspot Detection after Large Scale Disasters using Location Data from Smartphones: Case Study of Kumamoto Earthquake. In Proceeding of 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2016). November 3, 2016, California.

Yongli Zhang, Christoph F. Eick. 2016. ST-DCONTOUR: a serial, density-contour based spatio-temporal clustering approach to cluster location streams. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming*. ACM.