4-2011

Polygonal Spatial Clustering

Deepti Joshi
University of Nebraska-Lincoln, djoshi81@gmail.com

POLYGONAL SPATIAL CLUSTERING

by

Deepti Joshi

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professors Ashok Samal and Leen-Kiat Soh

Lincoln, Nebraska

May, 2011

POLYGONAL SPATIAL CLUSTERING

Deepti Joshi, Ph.D.

University of Nebraska, 2011

Adviser: Ashok Samal and Leen-Kiat Soh

Clustering, the process of grouping together similar objects, is a fundamental task in data mining to help perform knowledge discovery in large datasets. With the growing number of sensor networks, geospatial satellites, global positioning devices, and human networks tremendous amounts of spatio-temporal data that measure the state of the planet Earth are being collected every day. This large amount of spatio-temporal data has increased the need for efficient spatial data mining techniques. Furthermore, most of the anthropogenic objects in space are represented using polygons, for example – counties, census tracts, and watersheds. Therefore, it is important to develop data mining techniques specifically addressed to mining polygonal data. In this research we focus on clustering geospatial polygons with fixed space and time coordinates.

Polygonal datasets are more complex than point datasets because polygons have topological and directional properties that are not relevant to points, thus rendering most state-of-the-art point-based clustering techniques not readily applicable. We have addressed four important subproblems in polygonal clustering. (1) We have developed a dissimilarity function that integrates both non-spatial attributes and spatial structure and context of the polygons. (2) We have extended DBSCAN, the state-of-the-art density based clustering algorithm for point datasets, to polygonal datasets and further extended it to handle polygonal obstacles. (3) We have designed a suite of algorithms that incorporate user-defined constraints in the clustering process. (4) We have developed a spatio-temporal polygonal clustering algorithm that uniquely treats both space and time as first-class citizens, and developed an algorithm to analyze the movement patterns in the spatio-temporal polygonal clusters. In order to evaluate our algorithms we applied our algorithms on real-life datasets from several diverse domains to solve practical problems such as congressional redistricting, spatial epidemiology, crime mapping, and drought analysis. The results show that our algorithms are effective in finding spatially compact and conceptually coherent clusters.

ACKNOWLEDGEMENT

I would like to thank my advisors Dr. Ashok Samal and Dr. Leen-Kiat Soh for their continuous guidance and support throughout the course of my doctoral studies. They have helped me become the scholar that I am today. Without their patience and faith in me, I wouldn't have accomplished the goal of finishing this dissertation. I am also thankful for the guidance of my supervisory committee members: Dr. Peter Revesz, Dr. David Marx, and Dr. Xun-Hong Chen. I would like to thank the Department of Computer Science and Engineering at the University of Nebraska-Lincoln for providing me this great opportunity. Finally, I would like to thank my family, especially my parents Mr. Ravinder Joshi and Mrs. Neelum Joshi, and my husband Mr. Arpit Sharma for their continuous love and support.

I am also grateful to the National Science Foundation for funding my research. This dissertation is based upon work supported by the National Science Foundation under Grants No. 0219970 and 0535255.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Data Mining is an important, fascinating, and a very active field in Computer Science that has revolutionized many endeavors, and will play a central role in laying the foundation for next generation of major advances in many disciplines such as geography, biology, medicine, and social and political science. It is a field drawing on algorithm design, system building, statistical analysis, simulation, and visualization.

Within the vast domain of data mining, spatial and spatio-temporal data mining are important fields of research. Spatial data mining is the process of extracting potentially useful and previously unknown information from spatial datasets. Explosive growth and widespread use of spatial datasets by organizations such as the National Aeronautics and Space Administration, Census Bureau, Department of Commerce, and National Institute of Health (NIH) have necessitated the development of efficient and scalable algorithms to extract knowledge from these huge datasets (Shekhar & Zhang, Spatial Data Mining: Accomplishments and Research Needs (Keynote Speech), 2004). Spatial datasets are unique in that they store the spatial information, i.e. longitude and latitude, the surrogate variables for space, of every object. The normal principles of independence that are assumed in the general data mining algorithms no longer apply. On the other hand, principles such as Tobler's First Law of Geography – 'All things are related, but nearby things are more related than distant things (Tobler W. , 1979),' and spatial autocorrelation (Zhang, Huang, Shekhar, & Kumar, 2003) become increasingly important (Shekhar, Zhang, Huang, & Vatsavai, 2003). As a result, the complexity of the techniques required to analyze the spatial datasets increases significantly. Furthermore, the advances in this area are so rapid that the 2010 University Consortium for Geographic Information Science (UCGIS) Summer Assembly, a leading body in Geographic Information Science and Technology (GIS&T) was called to address the changes happening in GIS&T theory, technology and applications. In the top nine research priorities identified by UCGIS spatiotemporal representation and modeling was ranked first, and spatiotemporal dynamics was ranked fifth. Some other research priorities (unranked) that were identified included Volunteered Geographic Information, spatial analysis and modeling, geovisualization, and prediction (Prager, 2010).

Spatial clustering, one of the most fundamental tasks in spatial data mining, has been steadily gaining importance over the past decade (Han, Kamber, & Tung, Spatial clustering methods in data mining: A Survey, 2001). It is the process of the arranging spatial objects into groups known as clusters such that the objects within the same group are similar to each other but dissimilar from the objects in other groups. Several spatial clustering algorithms have been proposed in the literature; a survey is presented in (Han, Kamber, & Tung, Spatial clustering methods in data mining: A Survey, 2001). However, the focus of researchers so far has been on point datasets with the idea that any spatial object can be represented as a point. Although this approximation makes the problem more tractable, this approach does not work well for spatially extended objects. This is because point representation of spatially extended objects such as polygons results in significant loss of structural and topological information that is critical in many applications (e.g. congressional redistricting and watershed analysis). The problem of polygonal clustering has been overlooked in the past, and is the focus of our research.

In our research, we focus on *clustering spatially extended objects that can be represented as polygons*. It is important to devise mechanisms for clustering polygons because most objects in the geographic space are two dimensional and they are more accurately represented as two-dimensional polygons than one-dimensional points. Moreover, many applications require that the spatial objects be represented as polygons. The geographic space can be logically organized into polygons that are either natural or man-made units, for example, watersheds, counties, congressional districts, agro-eco zones, and natural resource districts. These as well as other domains can benefit from polygonal clustering algorithms. The resultant clusters can be used for classification, prediction, scientific analysis, decision making, or for simply map formation and visualization.

1.1 Applications

Most of the anthropogenic objects such as parks, administrative areas, market areas, buildings, and vehicles all lend themselves to a polygonal representation (Robertson, Nelson, Boots, & Wulder, 2007). Furthermore, the geographic space can be organized as polygons. For example, there are naturally formed polygons such as lakes, watersheds, rivers basins, and aquifers, or human-defined polygons such as states, counties, and census tracts.

In the geospatial domain, a central problem is organizing the space into regions for easier management and analysis. Often it becomes a problem of aggregating smaller regions into larger ones. This is fundamentally a polygonal clustering problem. For example, congressional redistricting is a problem that is revisited every 10 years in the United States. However until today, there is no proper method to automate the process and evade the issue of gerrymandering completely. Other examples of zone formation are school districts, police precincts, and electricity dispersion zones. Examples of other applications of polygonal clustering include, but are not limited to, watershed analysis, drought analysis, crime mapping, and spatial epidemiology.

1.2 Problem Description

Many applications in the geospatial domain require organizing the space into clusters of polygons that are spatially contiguous and compact. When polygons are represented as points, the clustering algorithms produce spatially disjoint clusters. This is because when a polygon is represented as a point, spatial and topological information such as the extent of boundary shared with another polygon is lost. Even in applications where spatial contiguity is not a factor, there is no appropriate point-based representation of a polygon that is embedded inside another polygon.

Structural complexity of the polygons and their distribution in space also induces additional challenges. For example, the size of the polygons in a dataset may be unbalanced, i.e. half are small, and the other half much bigger. The question is then: should the small and big polygons be treated equally? Another scenario may involve two or more polygons sharing one or more spatial object, for example, two or more counties sharing the same river. How should the relationship be defined among these polygons that share the same spatial object? Yet another example of such an issue is when two polygons are divided by a linear spatial object, such as a river or a mountain range. Does the presence of the linear spatial object decrease or increase the similarity between the two polygons? Finally, while in general, point datasets may contain noise or outliers, they are relatively uncommon in the polygon datasets. Therefore, most of the times, all the polygons present in a dataset need to be accurately clustered.

Furthermore, the problem of district and zone formation is particularly a difficult problem to solve. This problem, in the past, has been deemed as computationally too expensive to be automated (Altman, 2001). This problem and other regionalization problems can be formulated as polygonal clustering where the clusters must be spatially contiguous and compact. Representing polygons as points and applying the point-based clustering algorithms may result in clusters that are spatially disjoint, or clusters that meander all across space.

Finally, the temporal domain is ever present in any real-life application. Everything changes with time. Animals migrate from one place to another with changing weather conditions; people move from under-developed to developed places in the world; with the increased global warming, there are climatic shifts happening around the world (Ravelo, Andreasen, Lyle, Olivarez Lyle, & Wara, 2004). As a result, polygons that define most of these things also do not remain constant in space across time (Robertson, Nelson, Boots, & Wulder, 2007). Thus, it is natural that the polygonal clusters would also change their shape and location across time. Therefore, it is not only important to develop techniques to identify static spatial clusters, but also clusters that are dynamic in nature. Representing time as a first-class citizen in the spatio-temporal clustering problem is an important challenge that has been a struggle in geospatial research. Most of the past research performs spatial clustering at different snapshots in time and then compares the resulting clusters (Kalnis, Mamoulis, & Bakiras, 2005). Performing true 3-dimensional clustering in space and time is a challenge that needs to be addressed.

Thus the problem of polygonal clustering can be defined as: *given a set of geospatial polygons defined in both space and time, group the polygons into a set of clusters such that the polygons within the same cluster are similar to each other with respect to their spatial and non-spatial properties*.

1.3 Proposed Approach

In this research we have addressed several fundamental problems in polygonal spatial clustering. The basic principles used to solving these problems are:

1. *Spatial Extent*: Represent a polygon as a two dimensional entity with a set of vertices rather than only the centroid of the polygon in order to accurately represent the location of a polygon. Using the centroid representation of the polygon may lead to inaccurate distance computation between two polygons.
2. *Spatial Attributes*: Integrate the spatial attributes and structure of polygons into the clustering process. Spatial attributes include area, perimeter, minimum bounding rectangle, ratio of the principal axes, shared boundary length, neighboring polygons, etc. Another level of spatial attributes includes other spatial objects embedded within the polygons. For example in a county, other spatial objects (e.g. lakes) may be present that can be represented as polygons themselves.
3. *Spatial Relationships*: Take into consideration the binary relationships that may exist within the polygonal datasets. For example, two polygons sharing a linear feature such as

a river may exhibit similar properties, and thus be related to each other with respect to the river.

4. *Spatial Autocorrelation*: Guide the clustering process according to the principles that reflect the nature of the geographic space, e.g. spatial autocorrelation, spatial heterogeneity, and Tobler's First Law of Geography.
5. *Density Connectivity*: Extend the density-based connectivity concepts from points to polygons in order to perform density-based polygonal clustering.
6. *Spatial Constraints*: Improve the clustering process further by the addition of different types of user-defined constraints, e.g. hard or soft constraints, instance-level constraints or cluster-level constraints (Davidson & Ravi, Towards efficient and improved hierarchical clustering with instance and cluster level constraints, 2004).
7. *Time as a First Class Citizen*: Treat both space and time as equals in the clustering process in order to bridge the gap between the spatial and temporal dimensions, and detect dynamic clusters and their movement patterns across space and time.

1.4 Research Contributions

In this research, we have made four significant contributions to the state of the art in polygonal clustering. They are briefly summarized below.

1. *Dissimilarity of Geospatial Polygons*: We have developed a ***polygonal dissimilarity function*** (Joshi, Samal, & Soh, A Dissimilarity Function for Clustering Geospatial Polygons, 2009a), (Joshi, Samal, & Soh, A Dissimilarity Function for Complex Spatial Polygons, Under Review) that accurately computes the dissimilarity between two polygons by integrating both non-spatial attributes and spatial structure and context of the polygons.
2. *Density-Based Polygonal Clustering*: We have developed a *density-based clustering algorithm for polygons known as **P-DBSCAN*** (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b). P-DBSCAN extends DBSCAN (Ester, Kriegel, Sander, & Xu, 1996), the

state-of-the-art density based clustering algorithm for point datasets to polygonal datasets. We have further extended the algorithm *to cluster polygons in the presence of obstacles* ***(P-DBSCAN+)*** (Joshi, Samal, & Soh, Polygonal Spatial clustering in the Presence of Obstacles , Under Preparation).

3. *Polygonal Clustering with Constraints*: We have developed a *suite of* ***constraint-based polygonal spatial clustering (CPSC) algorithms*** (Joshi, Soh, & Samal, Redistricting Using Heuristic-Based Polygonal Clustering, 2009c), (Joshi, Soh, & Samal, Redistricting using Constrained Polygonal Clustering, Under Review) for clustering polygons under a given set of user-defined constraints. These algorithms provide a systematic approach for incorporating both hard and soft constraints, and holistically integrating them in the clustering process.
4. *Spatio-Temporal Polygonal Clustering*: We have developed a ***spatio-temporal polygonal clustering (STPC) algorithm*** (Joshi, Samal, & Soh, Detecting Spatio-Temporal Polygonal Clusters Treating Space and Time as First Class Citizens, Under Review) that uniquely treats both space and time as first-class citizens. Using this algorithm we are able to bridge the gap between the spatial and temporal dimensions, and overcome the bottleneck of snapshot approaches. Furthermore, in order to detect the dynamic changes that a cluster goes through in its lifetime, we have developed an algorithm known as ***Detecting Movements in Spatio-Temporal Clusters (DMSTC)*** (Joshi, Samal, & Soh, Analysis of Movement Patterns in Spatio-Temporal Polygonal Clusters, Under Preparation) that analyzes the movement patterns in spatio-temporal polygonal clusters.

1.5 Dissertation Overview

The structure of this dissertation is as follows. Chapter 2 describes the details of the polygonal dissimilarity function. We also present the results obtained by applying our dissimilarity function on a watershed dataset and county dataset. Chapter 3 presents the density-based clustering algorithm for polygons known as P-DBSCAN. We also show the application of P-DBSCAN on a county dataset in order to detect density-connected clusters of polygons. Chapter 4 details the density-based clustering algorithm for polygons in the presence of obstacles known as P-DBSCAN+. Followed by which we show the application of P-DBSCAN+ on a census tract data-set in the presence of obstacles such as rail-road tracks and rivers. Chapter 5 describes the suite of constraint-based clustering algorithms for polygons known as CPSC, CPSC* and CPSC*-PS. In this chapter we show the results for the congressional redistricting and school district formation applications. Chapter 6 presents the spatio-temporal polygonal clustering (STPC) algorithm. We show the results of the application of STPC for drought analysis, spatial epidemiology, and crime mapping applications. Chapter 7 presents the DMSTC algorithm that analyses the movement patterns of spatio-temporal clusters as they move from one time stamp to another. This chapter also shows the results of the analysis of the movement patterns of drought clusters, flu clusters, and crime clusters. Finally, in Chapter 8 we present a summary of our work, along with directions for future research.

Chapter 2: A Dissimilarity Function for Geospatial Polygons

2.1 Introduction

Explosive growth and widespread use of spatial datasets by organizations such as the space agencies worldwide, the census bureau, and healthcare agencies have led to the need of developing efficient and scalable algorithms to extract knowledge from these huge datasets (Shekhar & Zhang, 2004). Spatial datasets are unique in that they store the spatial information in the form of the longitude and latitude of every object. As a result, the complexity of the datasets increases. Unlike transactional data, principles such as Tobler's first law of geography – 'All things are related, but nearby things are more related than distant things (Tobler, 1979),' and spatial autocorrelation play significant role (Zhang et al, 2003) within the spatial datasets. As a result, the normal principles of independence that are assumed in the machine learning algorithms are not applicable to the spatial datasets.

Spatial data can further be divided into three different categories – point spatial datasets, linear spatial datasets, and polygonal spatial datasets. While points datasets are easily represented using their latitude and longitude, linear and polygonal datasets are much more complicated in nature (Pease note that the polygons referred to here are the same as regions (Cliff et al, 1975) or tessellations in space.) For example, the length of boundary shared between two polygons—which may be used to determine spatial proximity of the two polygons—is lost when polygons are represented as points. Moreover, for a concave shaped polygon, the centroid of the polygon may lie outside the boundary of the polygon. Thus, if one tries to spatially analyze polygons simply by representing them as points (typically their centroids) the result may not be accurate, and the underlying spatial structure is lost. Furthermore, when considering spatial polygons, there may be other spatial objects that lie within the polygons or may be shared by two or more polygons. For example, lakes, rivers, and even manmade structures such as highways lie within geospatial polygons such as counties and watersheds. There is no appropriate representation of this type of information when using the current state-of-the-art in spatial analysis. For example, if one were to perform watershed analysis – where watersheds are naturally formed polygons within the river basins – based on their relationship with a set of rivers, say, cutting through the watersheds, there is no current spatial analysis technique that would allow us to do so.

In this chapter we propose a new dissimilarity function called the *Polygonal Dissimilarity Function* (PDF) that comprehensively integrates both the spatial and the non-spatial attributes of a polygon to specifically consider the spatial structure and organization of the polygons. This is based on our earlier work presented in (Joshi et al. 2009b). We hypothesize that, in order to accurately represent polygons in the geospatial domain, the attributes of the polygons should accurately capture both its spatial structure (intrinsic to the polygon) and its spatial organization (extrinsic to the polygon) along with the non-spatial attributes of the polygons. The spatial structure of a polygon represented using a set of intrinsic attributes refers to the area covered by polygon, its location, its shape, etc. By taking the intrinsic attributes of the polygon into account we can find out, for example, the extent of the boundary shared by two polygons, the information as mentioned before that is lost by representing the polygon as a point. On the other hand, the spatial organization of a polygon represented using a set of extrinsic attributes refers to the topological relationship between the polygon and its neighboring polygons within the dataset as well as other spatial objects present within a polygon itself. Measuring the extrinsic attributes of the polygons would thus allow us to take into account for example the spatial distributedness of other spatial objects present within the polygons, giving us another perspective on the similarity between polygons. Using this representation of the polygons, we define PDF as a weighted function of the distance between two polygons in the different attribute spaces. In other words, PDF is a combination of a number of distance functions each pertaining to a different class of attributes describing a polygon. Furthermore, the weights in the dissimilarity function allow the users to customize their use of PDF based on the significance of the attributes in their application domain. For example, in order to find the similar lakes based on their topological relationship—such as "adjacent to"— with watersheds, one would assign a greater weight to the spatial distance that measures topological relationships. On the other hand, in order to discover the lakes high in nitrogen content, a higher weight must be assigned to the non-spatial attributes. In Section 2.3 we describe the distance functions for the underlying attributes of the polygons along with the guidelines for combining them effectively.

Our novel dissimilarity function can be used in a variety of problems where distance or similarity plays a central role. Examples of such application areas include – clustering of geospatial polygons, training of an instance-based learning system, prediction and trend analysis, etc. Clustering, a common data mining task is a prime application for a dissimilarity function since it is based on separation of dissimilar objects, and grouping of similar objects. Other applications, such as region growing in which objects are ranked based on degree of similarity to their neighboring polygon, require a function that orders polygons in increasing similarity. Most distance functions used in polygonal clustering or regionalization fail to comprehensively treat all the spatial attributes (see Section 2.2 for an overview of the most commonly used distance functions for polygons) due to the inadequate representation of structural (intrinsic) and topological (extrinsic) information contained in the polygons. This leads to inaccuracy in the computed results. It is our hypothesis that the use of PDF will lead to more accurate comparison of polygons.

In order to evaluate our dissimilarity function we first compare and contrast it with other distance functions proposed in literature that also use both spatial and non-spatial attributes. In particular, we compare our algorithms to the distance functions proposed by Webster and Burrough (1972), Cliff *et al.* (1975), and Perruchet (1983). These distance functions have been described in Section 2.2, and the comparative analysis has been presented in Section 2.4. Next, we specifically investigate the effectiveness of our dissimilarity function in spatial clustering since distance based functions play a central role in this application. We have applied our dissimilarity function to the *k*-medoids clustering algorithm to cluster geospatial regions represented as polygons in two different domains with diverse characteristics – namely, environmental analysis using watersheds and political applications using counties. Our results show that PDF outperforms other distance functions in ranking the similarity between polygons, and results in the maximum range between the pair-wise distances computed. Furthermore, our results for the clustering application show that with the use of the intrinsic and extrinsic spatial attributes of the polygons along with the non-spatial attributes results in more cohesive clusters.

Finally, we use the term “dissimilarity” instead of “distance” because our dissimilarity function does not satisfy the symmetry and triangular inequality properties of distance metric (Arkhangel'skii & Pontryagin, 1990).

2.2 Related Work

In this section we present an overview of the various distance functions proposed in literature for measuring the distance between two polygons along with the problems associated with their use. Polygons in general can be concave or convex, small or large, elongated or compact. Furthermore, completely disjoint polygons can have overlapping bounding boxes; adjacent polygons can share a single point, a segment on the boundary or even multiple segments. Based on these properties of the polygons the following distance functions have been proposed.

Centroid Distance. One way to approximate polygon objects is to represent each object by a representative point, such as the centroid of each object, and then find the distance between the centroids of the polygons. However, this approach is generally not effective since the objects may have very different sizes and shapes. For instance, a rectangular building may have a size of 500 square meters, whereas a lake may have a size of 300,000 square meters with irregular elongated shape. Simply representing each of these objects by its centroid, or any single point, does not take into the account the extents of the polygons. Another problem with this approach is that the centroid may not be inside the polygon (e.g., for some concave objects) and may indeed be inside another object.

Minimum Bounding Rectangle Distance. There is a large body of work in shape analysis (e.g., Gardoll 2000; Shapiro & Stockman, 2001). For example, the minimum bounding rectangle (MBR) of a polygon can be used as a first-order approximation of the shape and orientation of the polygon: it is the smallest rectangle that encloses an object. Distance of two polygons can be measured by finding the distance between the centers of their respective MBRs. However, many of the same problems described for centroid-based distances remain. For example, the center of the MBR of a polygon may not fall within the polygon, or the MBRs of two polygons may overlap.

Separation distance. The distance between a point P and a line L is defined by the perpendicular distance, between the point and the line, i.e., $min\{d(P,Q)|Q \text{ is a point on } L\}$. Thus, given two polygons *A* and *B*, we can define the distance between these two polygons to be the minimum distance between any pair of points in *A* and *B*, i.e., $min\{d(P,Q)|P,Q \text{ are points in } A,B \text{ respectively}\}$. This distance is called the separation distance (e.g., distance between polygons P_1 and P_3 as shown in Figure 1 and is exactly the same as the minimum distance between any pair of points on the boundaries of *A* and *B* (Dobkin & Kirkpatrick, 1985).

However, if two polygons intersect or share boundaries or even a point, their separation distance is zero. This definition of distance is quite unsatisfactory for geospatial applications, e.g., the distances between P_1 and P_2 and between P_2 and P_3 , as shown in Figure 1. The separation distance between two adjacent polygons will always be zero and is an inappropriate measure since all polygons will have shared boundaries with their neighbors. The transitive relationship in terms of separation distance does not hold: in Figure 1, for example, the separation distance between P_1 and P_3 is non-zero, even though each has a zero separation distance with P_2.

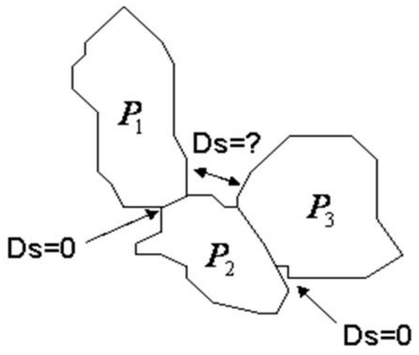
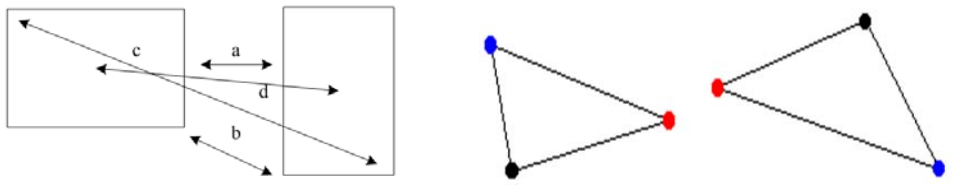


Figure 1: Separation distance where the transitive relation does not hold.

Min-Max Distance. Another way to measure the distance between polygons is to find the minimum or maximum distance between each pair of vertices of the polygons. However, this method either overestimates or underestimates the true distance between two polygons as shown in Figure 2(a). It shows the separation distance (a), the minimum distance between vertices (b), the maximum distance between vertices (c), and the distance between the centroids (d). It is clear that both b and c do not match the intuitive notion of the distance between the two polygons. If we only consider the minimum or the maximum distance between vertices, we overlook the shape of the polygons as shown in Figure 2(b), where the shortest and longest distances between any pair of vertices are shown in red and blue, respectively. Clearly, these distances are independent of the shape of the polygons, i.e. many polygons with different shapes can have the same distance as long as we maintain the two extreme (minimum or maximum) points in the two polygons. Hence these are inappropriate as distance measures.



(a) (b)

Figure 2: Minimum and maximum distance between vertices.

Hausdorff distance. The Hausdorff distance between two sets of points (Rote, 1991) is defined as the maximum distance of points in one set to the nearest point in the other set. Formally, the Hausdorff distance from set A to set B is defined as:

$$h(A,B) = \max_{a\in A}\left(\min_{b\in B} d(a,b)\right)$$

where a and b are points of sets A and B, respectively, and $d(a, b)$ is any distance metric between the two points a and b; for simplicity, we can take $d(a, b)$ as the Euclidian distance between a and b . If the boundaries of the polygons P_i and P_j are represented by two sets of points A and B, respectively, we can use this as a distance measure between two polygons.

$$D_h\left(P_i, P_j\right) = \max\left(h(A,B), h(B,A)\right)$$

Figure 3 presents a comparison between the Centroid distance and Hausdorff distance of two polygons. For convex polygons the Hausdorff distance, defined on the set of vertices of polygons, usually gives as good an estimate of distance as the Centroid distance. However, using the centroids to measure the distance between two polygons may not give us the “true” distance for concave polygons. As shown in Figure 3, the Centroid distance D_c may underestimate or overestimate the exact distance when the centroid of a concave polygon falls outside the polygon. The Hausdorff distance, D_h, defined on the two sets of vertices of polygons, gives a more accurate measurement.

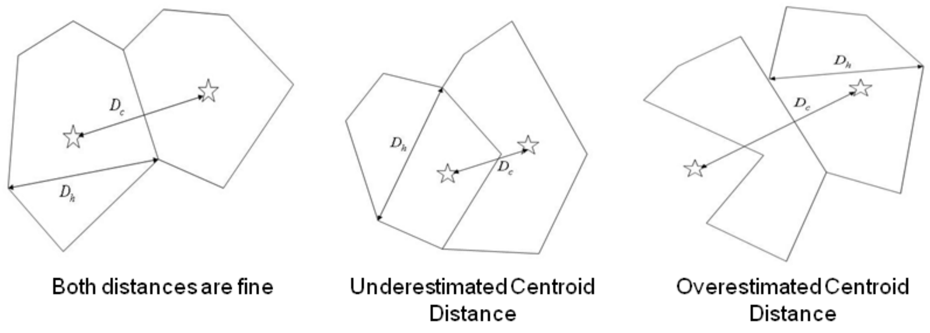

D_c
D_h
Both distances are fine
D_h
D_c
Underestimated Centroid Distance
D_h
D_c
Overestimated Centroid Distance

Figure 3: Comparison of Hausdorff distance with centroid distance.

Fréchet Distance. In order to measure the distance between polygons based on their shape, Fréchet distance (Buchin, Buchin, & Wenk, 2006) is considered to be more appropriate than Hausdorff distance (Rote, 1991). An intuitive definition of the Fréchet distance is to imagine that a dog and its handler are walking on their respective polygon boundaries. Both can control their speed but can only go forward. The Fréchet distance of these two polygon boundaries is the minimal length of any leash necessary for the dog and the handler to move from the starting points of the two curves to their respective endpoints. It is formally defined below:

Let *f, g* be parameterizations of curves or polygons, i.e., continuous functions

$$f, g : [0,1]^k \rightarrow R^d, k \in \{1,2\}, d \leq k$$

Then their *Fréchet distance (D_F)* is

$$D_F(f, g) = \inf_{\sigma:[0,1]^k} \max_{t \in [0,1]^k} \| f(t) - g(\sigma(t)) \|$$

where the *re-parameterization σ* ranges over all *orientation preserving* homeomorphisms.

It is important to note that Fréchet distance is used only for shape matching. It does not measure the geographic distance between two polygons in the geospatial applications for instance. For such purposes Hausdorff distance is more appropriate as shown in Figure 3.

In addition to the distance functions defined above, several ways to combine geographical distances and non-geographical dissimilarities into a single pair-wise similarity value have been proposed in literature. Webster and Burrough (1972), Cliff *et al.* (1975), and Perruchet (1983) proposed different multiplicative and additive forms to combine such elements. These are defined below:

WB Distance. Webster and Burrough (1972) proposed to compute the dissimilarity between pairs of polygons using the 'Canberra metric'. The Canberra metric between the i^{th} and the j^{th} sites is computed as follows:

$$D_{ij} = \sum_{k=1}^{p} \left[\frac{|g_{ik} - g_{jk}|}{g_{ik} + g_{jk}} \right] / p$$

Where g_{ik} and g_{jk} are the values of the k^{th} property for the i^{th} and j^{th} polygons respectively and p is the number of properties. They further proposed to add the geographic distance between the sites to the Canberra metric coefficient as follows:

$$D_{WB} = \frac{D_{ij} + \frac{d_{ij}}{d_{\max}} \times w}{1 + w}$$

Where D_{ij} is the Canberra metric between polygons *i* and *j*, d_{ij} is the geographic distance between the polygons *i* and *j*, $d_{\max}$ is the distance between the most distant pair of polygons, and w is a weighting factor.

CXY Distance. Cliff *et al*. (1975) propose a combined distance metric (D_{cliff}) to measure the distance between two polygons *i* and *j* as:

$$D_{cliff} = \lambda d_{ij} + (1 - \lambda) t_{ij}$$

where d_{ij} is some distance metric that measures the spatial separation between the i^{th} and j^{th} regions, t_{ij} is the distance metric that measures the distance between the non-spatial attributes of the two regions, and λ represents a weighing constant $(0 \le \lambda \le 1)$. $\lambda = 0$, represents a purely non-spatial strategy, and $\lambda = 1$ represents a purely spatial strategy. $\lambda = 0.33$ and $\lambda = 0.66$ signify a mixed strategy which has been shown by the authors to yield intermediate results with an average efficiency about twenty percent greater than that of the extremes.

PXY Distance. Perruchet (1983) defines the aggregation index of dissimilarity, D_P, between two polygons *i* and *j* as follows:

$$D_P(i, j) = f(\delta(i, j), d(i, j))$$

where $f(x,y) = xy$, $d(i,j)$ is the geographic distance between the two polygons and is computed using the Euclidean distance function, and $\delta(i,j)$ is the aggregation index defined as the dissimilarity between the polygons based on their non-spatial attributes. An example of $\delta(i,j)$ is given as:

$$\delta(i,j) = \frac{\mu_i \mu_j}{\mu_i + \mu_j} \left\| v_i - v_j \right\|^2$$

where μ_i is the mass of *i*, and v_i is the representation of *i* in the descriptor space.

In summary, all the distance functions defined above focus on one or two aspects (distance and/or shape) of polygons. Our representation of a polygon includes their structural and organizational properties which are fundamentally different, and thus need to be treated differently. These properties are not incorporated in any of the functions proposed in literature in a comprehensive manner. This serves as the motivation of our work to define a comprehensive dissimilarity function that effectively unifies the distance functions for each type of attribute of a polygon.

2.3 Dissimilarity Function for Geospatial Polygons

Consider a set of polygons $P = \{P_1, P_2, ..., P_n\}$ where each polygon P_i is defined by a set of spatial and non-spatial attributes.

The *non-spatial attributes* of a polygon include all the attributes of the polygon that are independent of the spatial location of the polygon. Examples of non-spatial attributes are – population, average income, number of hospitals, number of major cities, etc.

The *spatial attributes* of a polygon can be further divided into two categories: 1) intrinsic and 2) extrinsic. The *intrinsic* attributes describe the geometric properties of the polygon without any contextual information in a domain independent way. Examples of intrinsic attributes include location, shape, area, aspect ratio, etc. The location of the polygon is represented as a set of vertices, specified in some spatial coordinate frame.

The *extrinsic* attributes encompass the various spatial objects that may exist within a polygon, or may be shared by two or more polygons, which may however be defined independent of the polygon. Thus, the extrinsic attributes represent the elements that are either embedded into or intersect with the polygon. These elements exist independently of the polygon, but share the geographic space with it in some fashion. There can be three classes of spatial objects: point, linear and areal. Examples of point spatial objects include buildings, shopping complexes, etc. Examples of linear spatial objects include rivers, roads, and mountain ranges. Examples of areal objects include reservoirs, crop areas, forests, and large lakes.

Given two polygons, P_i and P_j, the ***Polygonal Dissimilarity Function (PDF)*** that measures the distance between two polygons in all the attribute spaces described above is defined as follows:

$$D_{PDF}(P_i, P_j) = f(d_{ns}(P_i, P_j), d_s(P_i, P_j)) \qquad (1)$$

where d_{ns} is a function that computes the distance between two polygons based on the non-spatial attributes – see Equation 3, and d_s is a function that computes the distance based on the spatial attributes – see Equation 4.

The function f in Equation 1 can be any non-spatial function that combines the two distances. We use a weighted sum that can easily adjust the contribution (i.e., the weight) of both the distances.

$$D_{PDF}(P_i, P_j) = w_{ns} d_{ns}(P_i, P_j) + w_s d_s(P_i, P_j) \qquad (2)$$

where $w_{ns} + w_s = 1$.

The weights w_{ns} and w_s are domain dependent, i.e. they should be tuned for the applications using experiential or expert knowledge. Therefore, we cannot explicitly assign them any fixed values. These weights play an important role in defining the contribution of the different types of attributes. For example in a clustering application of our dissimilarity function, if we are interested in clustering regions based on the density of population, and do not care that the regions should be spatially contiguous, a higher weight may be assigned to the non-spatial attributes. On the other hand, if we want the clusters to be spatially contiguous, a higher weight must be assigned to the spatial attributes.

2.3.1 Distance between Non-Spatial Attributes

The distance between the polygons in the non-spatial attribute space (d_{ns}), can be defined using any distance measure such as the Euclidean distance function or the Manhattan distance function. We use the standard Euclidean distance as our distance measure as shown in Equation 3.

$$d_{ns}(P_i, P_j) = \sqrt{\sum_{k=1}^{m} (g_{ik} - g_{jk})^2} \qquad (3)$$

where g_{ik} and g_{jk} represent the k^{th} non-spatial attribute of polygons P_i and P_j respectively, and m is the total number of non-spatial attributes. Please note that all the non-spatial attributes must be represented as ordered numerical attributes so that they can be integrated together. Furthermore, all the attributes must be normalized before the computation of the distance. The normalization can be performed by dividing all the values in the dataset by the largest value in the dataset (Han & Kamber, 2006). We assign an equal weight to all the non-spatial attributes. However, if desired, different weights may be assigned to the various non-spatial attributes. In this case, the equation for the distance function for non-spatial attributes will be as follows:

$$d_{ns}(P_i, P_j) = \sqrt{\sum_{k=1}^{m} w_k (g_{ik} - g_{jk})^2} \qquad (3\text{-}1)$$

2.3.2 Distance between Spatial Attributes

The distance between the polygons based on their spatial attributes (d_s) is defined as a function of the distance between their intrinsic spatial attributes (d_{ins}) and their extrinsic spatial attributes (d_{exs}) as reflected in Equation 4. The function d_{ins} is defined in Equation 6, and the function d_{exs} is defined in Equation 15.

$$d_s(P_i, P_j) = w_{ins} d_{ins}(P_i, P_j) + w_{exs} d_{exs}(P_i, P_j) \tag{4}$$

where $w_{ins} + w_{exs} = 1$.

2.3.2.1 Distance between Intrinsic Attributes

Among the intrinsic attributes of polygons, location is the most important. The location of a polygon is defined as a vector of its vertices. Intuitively, we expect the distance between two polygons with shared boundaries to be shorter than the distance between two polygons that do not have a common border. This is based on the assumption that two regions that share a boundary are closer than two regions—with everything else being equal—that do not, an assumption that has been used in domains dealing with spatial data such as image processing and structural organization (Jiao & Liu, 2008). The importance of geographic distance and the shared boundary length between two regions in various political applications have been demonstrated in (Furlong & Gleditsch, 2003).

The Hausdorff distance function as defined in Section 2.1 is a suitable distance function to measure the distance between the vertices of two polygons as it neither under-estimates nor over-estimates the distance between two polygons. However, the standard Hausdorff distance is defined on the set of points and does not incorporate any shared boundary. In order to incorporate this, we define a new distance measure, called *boundary adjusted Hausdorff distance* that is inversely proportional to the length of the shared boundary between two polygons P_i and P_j as follows:

$$d_{hs}(P_i, P_j) = \left(1 - \frac{2s_{ij}}{s_i + s_j}\right) \times d_h(P_i, P_j) \qquad (5)$$

where d_h is the original standard Hausdorff distance, s_i and s_j are the perimeter lengths of polygons P_i and P_j, respectively, and s_{ij} is the length of their shared boundary. This distance, d_{hs}, is smaller than the standard Hausdorff distance when two polygons have shared boundary, and becomes the standard Hausdorff distance when two polygons have no shared boundary, i.e., when s_{ij} = 0. We use twice the shared distance in the definition to balance the effect of the denominator.

Other than location, for the other intrinsic attributes, we compute the Euclidean distance between the individual attributes of the polygons in order to measure the distance between the polygons. Finally, the distance between polygons P_i and P_j based on their intrinsic attributes (d_{ins}) is defined as:

$$d_{ins}(P_i, P_j) = w_{hs} d_{hs}(P_i, P_j) + w_{st}\sqrt{\sum_{k=1}^{r}(t_{ik} - t_{jk})^2} \qquad (6)$$

where t_{ik} and t_{jk} represent the k^{th} structural attribute of polygons P_i and P_j respectively, and r is the total number of structural attributes, w_{hs} represents the weight assigned to the modified Hausdorff distance function, w_{st} is the weight assigned to the remaining intrinsic spatial attributes, and $w_{hs} + w_{st} = 1$.

2.3.2.2 Distance between Extrinsic Attributes

Extrinsic attributes incorporate the spatial objects present within the polygons or shared by two or more polygons. Given below is a framework that is used for defining the distance between two polygons based on their extrinsic attributes. The distance is based on the following properties of the various spatial objects with respect to the polygon – 1) *density*, 2) *extent* (the area covered by the object within the polygon), 3) *spatial distribution*, 4) *topology* and 5) *direction*.

The density, extent and distribution of a spatial object within a polygon are indicative of the underlying forces (e.g. climate or other biological or geophysical or chemical) which influence the polygon. In the geospatial domain for example, the presence of clusters of oak trees in two polygons is indicative of similar soil and/or climate regime, and therefore both the polygons are likely to be more similar to each other. Therefore two polygons with similar object density and distribution are more likely to be similar. The topology of spatial objects, on the other hand, especially of linear spatial objects, is important as it captures the binary relationship between the polygons with respect to other spatial objects. For example, a physical barrier between the polygons (e.g., a mountain range) can potentially increase the physical distance between the polygons, and hence discourage the polygons to be clustered together.

Due the wide differences in their construction, e.g. an areal object extends over a large area, whereas a point object is simply a single point within the polygon, not all the different aspects mentioned above are applicable to every type of spatial object. Table 1 lists the different types of characteristics applicable to the different types of spatial objects.

In Table 1, n is the number of times the spatial object occurs within the polygon, $|A|$ is the total area of the polygon, $|a_i|$ is the total extent of the areal object i within the polygon, and z_i is the test statistic obtained from the Mean Nearest Neighbor test for complete spatial randomness (CSR) (Donnelly 1978), and N/A stands for not applicable. Next, we define the functions that are used to find the distance between two polygons on the basis of the above mentioned properties of the spatial objects present within the polygons.

Table 1: Different characteristics of spatial object attributes

	Density	Extent	Distribution	Topology	Direction
Linear Object	$dn = \frac{n}{\lvert A \rvert}$	N/A	z_i	Defined below	Defined below
Areal Object	$dn = \frac{n}{\lvert A \rvert}$	$e = \frac{\lvert a_i \rvert}{\lvert A \rvert}$	z_i	Defined below	Defined below
Point Object	$dn = \frac{n}{\lvert A \rvert}$	N/A	z_i	N/A	N/A

Density and Extent. Density is the number of times an object occurs within a polygon divided by the area of the polygon. Extent is the total area covered by the object within the polygon. We measure the distance between two polygons on the basis of the density of the objects using Equation 7, and on the basis of extent using Equation 8.

$$d_{density} = \frac{\lvert dn_i - dn_j \rvert}{\max(dn_i, dn_j)} \tag{7}$$

where dn_i is the density of point object *m* in polygon P_i, dn_j is the density of point object *m* in polygon P_j.

$$d_{extent} = \frac{\lvert e_i - e_j \rvert}{\max(e_i, e_j)} \tag{8}$$

where e_i is the total extent of an areal object within polygon P_i, e_j is the total extent of the areal object within polygon P_j.

Distribution. The spatial distribution of an object is measured using the Mean Nearest Neighbor test for complete spatial randomness (CSR) (Donnelly 1978). The statistic produced as the output of this test is a fair indicator of the presence of *aggregation*, *regularity* or *randomness* of events located within a polygon. This information about the polygons helps us in identifying the polygons that have a similar underlying structure.

Please note that the spatial distribution test is only applicable for point data set. Therefore, in order to measure the distribution of areal objects, some methodology needs to be followed to represent areal objects as a set of points. While more complicated methods can be devised for this purpose, as the areal objects present within the polygons are an order smaller in magnitude, for simplification purposes we represent each areal object by its centroid. To measure the distribution of linear objects, we take a fixed number of points from each linear object, and use these points for the spatial randomness test. We measure the distance between two polygons on the basis of the distribution of the spatial objects using equation 9.

$$d_{distribution} = \frac{|z_i - z_j|}{\max(z_i, z_j)} \qquad (9)$$

where z_i is the distribution of the point object *m* in polygon P_i, and z_j is the distribution of the point object *m* in polygon P_j.

Topology. Relationships between a pair of spatial objects (points, lines, and regions) can be characterized as topological relations that describe how two such objects interact in a 2D space. The 4-intersection model, and the 9-intersection model (Egenhofer & Franzosa, 1994) describe an object as its interior, boundary, and exterior. The relationship between two objects is then based on the intersection of their interior, exterior or boundary. The topological relationship between two objects helps us in computing the distance function in between the two objects – two objects with similar topology are more likely to be similar than two objects with different topology. Here we provide an extension of the framework proposed by Egenhofer and his colleagues (Egenhofer & Franzosa, 1994), (Egenhofer & Mark, 1995), (Egenhofer, Clementini, & Felice, 1994) so that the topological relationship between two spatial objects can be defined with reference to a third spatial object. The *topology* of a polygon with respect to the linear objects is defined as follows.

A polygon (A) can be divided into three segments – boundary (∂A), interior (A°), and exterior (A^{-}). A linear feature (l) may intersect the boundary, the interior, or the exterior of the polygon. Furthermore, the polygons may lie either on the same side of the linear feature, or they may be on opposite sides of the linear feature. Table 2 illustrates the different scenarios that may arise and define the topological relationships between the two polygons based on a linear feature. These scenarios are also demonstrated in Figure 4. Once the relationship between two polygons with respect to a linear feature is determined, the distance between the two polygons is computed on the basis of the following two rules: 1) If the linear feature intersects the interior of both the polygons, then the distance between them is the smallest. 2) If the linear feature intersects only the exterior of both the polygons, then the distance is the largest.

Table 2: Different possible scenarios based on topological relationship of a linear feature (l) with two polygons (A and B)

		∂A	A°	A^{-}	∂B	B°	B^{-}	**Figure**
Scenario 1	l	X			X			Figure 4(a) & 4(b)
Scenario 2	l	X				X		Figure 4(c)
Scenario 3	l	X					X	Figure 4(d) & 4(e)
Scenario 4	l		X		X			Figure 4(c)
Scenario 5	l		X			X		Figure 4(f)
Scenario 6	l		X				X	Figure 4(g)
Scenario 7	l			X	X			Figure 4(d) & 4(e)
Scenario 8	l			X		X		Figure 4(g)
Scenario 9	l			X			X	Figure 4(h) & 4(i)

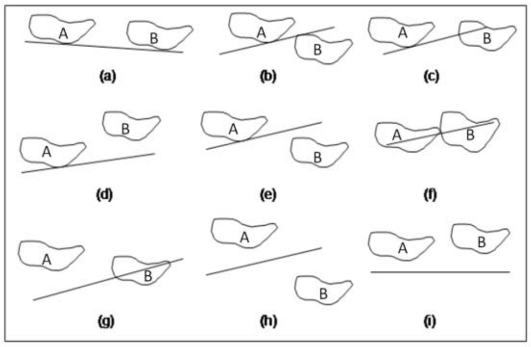


Figure 4: Topological relationship between two polygons based on a linear feature – linear feature may intersect the interior, exterior or the boundary of a polygon.

We generalize the distance function between a polygon and the linear feature based on the topological relationship as follows.

$$d(P_i, l) = \begin{cases} \beta & \text{if } P_i^\circ \cap l \neq \phi \quad (interior) \\ \alpha & \text{if } \partial P_i \cap l \neq \phi \quad (boundary) \\ \alpha + f(P_i^-, l) & \text{if } P_i^- \cap l \neq \phi \quad (exterior) \end{cases} \qquad (10)$$

where P_i is any polygon, and l is any linear feature, $0 \leq \beta < \alpha \leq 1$, and $f(P_i^-, l) =$ Nearest distance of P_i to l. Here β is defined as the constant minimum distance that any polygon will have to a linear feature that intersects its interior and α is defined as the constant distance that any polygon will have to a linear feature that intersects its boundary.

The topology of a polygon with respect to an areal object will follow the same design as presented for the linear object. The scenarios illustrated in Table 2 and Figure 5 can be extended for areal objects by replacing the linear feature with an areal object. For example, if we take into consideration underground aquifers which are shared by two watersheds, then the areal object (underground aquifers) intersects the interior of both the polygons P_i and P_j (the two watersheds). The distance, based on the topology of the polygon with respect to the areal object, between the areal object P_i will be equal to β and the distance of polygon P_j to the areal object will also be β. Similarly, the rest of the cases can be extended from the linear objects to the areal spatial objects.

Direction. The linear and areal spatial objects may also impose directional constraints on the polygons, i.e. the polygons may be on the same side of the linear feature, or on opposite sides. Furthermore, a linear or an areal feature present between two polygons may increase or decrease the distance between the polygons. Based on these two factors, the distance between two polygons P_i and P_j based on the linear feature *l*, is given by the following function:

$$d(P_i, P_j \mid l) = \lfloor oppg(P_i, P_j, l) + \gamma \rfloor \times opps(P_i, P_j, l) + \beta \qquad (11)$$

where $oppg(P_i, P_j, l)$ is 1 if the linear feature l is defined to be opposing (i.e. the presence of the linear feature makes the polygons dissimilar), and 0 otherwise; γ is defined as a constant that ensures that the linear feature which is opposing in nature increases the distance between two polygons to an extent so that they are not categorized as similar polygons; and $opps(P_i, P_j, l)$ is an indicator of the location of the polygons with respect to the linear feature. It has a value of 1 if the polygons are on the opposite sides of the linear feature, or 0 if the polygons are on the same side.

It should be noted that these values are also domain dependent. If the domain is such where the polygons are considered to be closer to each other if they are on opposite sides of the linear feature rather than on the same side, then the values defined for $opps(P_i, P_j, l)$ can be reversed. This same argument also applies to $oppg(P_i, P_j, l)$. Similarly, we can apply the above equation for an areal object when it is shared by two polygons.

Due to the difference in the characteristics of the three types of spatial objects, we treat them separately, and define three different functions, d_τ, d_φ and d_ω to compute the distance for point, linear, and areal objects defined in Equations 12, 13 and 14 respectively.

The distance based on *spatial point objects* is:

$$d_\tau(P_i, P_j) = \frac{1}{k}\sum_{m=1}^{k}(d_{density} + d_{\text{distribution}}) \qquad (12)$$

where k is the total number of different point objects present within both polygons P_i and P_j ,

The distance based on *spatial linear objects* is:

$$d_\varphi(P_i, P_j) = \frac{1}{l}\sum_{m=0}^{l}(d_{density} + d_{distribution} + [d(P_i, m) + d(P_j, m)] \times [1 + d(P_i, P_j \mid m)]) \qquad (13)$$

where l is the total number of different linear objects present within both polygons P_i and P_j ,

Note: There may exist a scenario where the linear feature m is exterior to both the polygons, and the nearest distance between m and both the polygons is fairly large. In this case, we propose to not include this linear feature while computing the distance between the two polygons based on the set of the linear features.

The distance based on *spatial areal objects* is:

$$d_{\omega}(P_i, P_j) = \frac{1}{a}(\sum_{m=1}^{a}(d_{density} + d_{distribution} + d_{extent} + [d(P_i, m) + d(P_j, m)] \times [1 + d(P_i, P_j \mid m)])) \quad (14)$$

where a is the total number of different types of areal objects within polygons P_i and P_j, is the extent of the areal.

Finally, we compute a weighted sum of the above three distances (d_{τ}, d_{φ} and d_{ω}) to derive the overall distance between two polygons P_i and P_j based on the organizational attributes as follows:

$$d_{exs}(P_i, P_j) = w_{\tau} d_{\tau}(P_i, P_j) + w_{\varphi} d_{\varphi}(P_i, P_j) + w_{\omega} d_{\omega}(P_i, P_j) \quad (15)$$

where $w_{\tau}, w_{\varphi}, w_{\omega}$, are the weights associated with the three spatial object types and $w_{\tau} + w_{\varphi} + w_{\omega} = 1$. These weights provide flexibility for the domain expert to emphasize any set of the spatial object features.

2.4 Experimental Analysis

In order to evaluate our polygonal dissimilarity function, we first compare and contrast it with other distance functions proposed in literature that also use both spatial and non-spatial attributes. In particular, we compare our algorithms to the distance functions proposed by Webster and Burrough (1972), Cliff *et al.* (1975), and Perruchet (1983). For this study we have made use of a subset of the census block groups from the city of Lincoln, NE along with the number of liquor licenses assigned to each census block group. The goal is to study the differences in the pair-wise distances computed for the polygons (census block groups) using the various distance functions.

Next, we specifically investigate the effectiveness of our dissimilarity function in spatial clustering since distance based functions play a central role in this application. We have applied our dissimilarity function to the *k*-medoids clustering algorithm to cluster geospatial regions represented as polygons in two different domains with diverse characteristics. We first apply our dissimilarity function to the hydrology domain where we examine the formation of clusters of watersheds. In hydrology watersheds are polygons that serve as the basic unit for analysis. For example, watersheds are often clustered together to perform frequency analysis of floods, or determine regional trends (Rao and Srinivas 2005). In a second experiment, we form clusters of counties that are often used for the organization of higher level political or management districts. In contrast to watersheds which represent natural units of area (polygons) with no defined geometric shape, counties are man-made polygons that have more regular geometric shape and spatial relationships.

2.4.1 Comparative Analysis

In this section we compare the performance of our algorithm with three distance functions that make use of both the spatial and non-spatial attributes, namely, the WB Distance, the CXY Distance and PXY Distance. These are described in Section 2.2. We use a set of six polygons which are census blocks in the city of Lincoln, NE (USA). For non-spatial attribute, we use the locations of liquor licenses within the census blocks. The census blocks and the sites for liquor licenses are shown in Figure 5.

The WB distance function is computed using – 1) the Canberra Metric on the number of liquor licenses, 2) the Euclidean distance between the centroids of the polygons, and 3) $w = 0.66$. The CXY distance is computed by: 1) taking the normalized Euclidean distance between the centroids of the polygons as the spatial distance function, 2) the normalized Euclidean distance between the number of liquor licenses, and 3) $\lambda = 0.66$. The PXY distance function is computed as the product of the spatial distance between the centroids of the polygons and the non-spatial distance using the metric defined in Section 2.2 (Perruchet 1983). We further compute a revised version of the PXY distance function where we force the mass of any attribute being considered to be at least 0.01. The results of this version of PXY distance are listed in Table under PXY'.

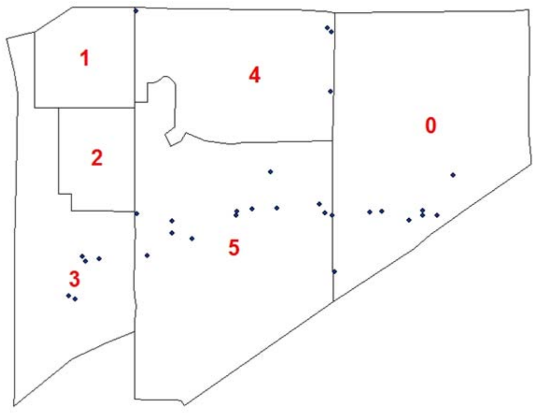


Figure 5: A set of census blocks in Lincoln, NE and the locations of the sites for liquor licenses.

Finally, we compute our proposed PDF using the – 1) non-spatial distance function computed as the normalized Euclidean distance between the number of liquor licenses, 2) intrinsic spatial distance function computed using our modified Hausdorff distance function, and 3) extrinsic spatial distance function computed using the density and distribution of the liquor license locations within each polygon. Further, the spatial attributes are assigned an overall weight of 0.66, and non-spatial attributes are assigned a weight of 0.34. Within the spatial attributes, intrinsic spatial attributes are assigned a weight of 0.5, and extrinsic spatial attributes are assigned a weight of 0.5. The number of liquor licenses within each block (NoLL1 and NoLL2) and the distances between each pair of polygons (P1 and P2) using the aforementioned functions for the census blocks shown in Figure 5 are presented in Table 3.

Observing the results obtained, we find that PXY distance function fails to compute the distance between two polygons when the mass of an attribute of either of the two polygons is 0. Moreover, it favors small attribute values, and heavily penalizes the attributes with large values.

This can be seen in the low distance computed for polygons with fewer liquor licenses, and a very large distance computed for the polygons with more liquor licenses.

Table 3: Statistics and the distances between polygons using different distance functions (WB Distance, CXY Distance, PXY Distance, PXY' Distance and PDF).

P1	P2	NoLL1	NoLL2	WB	CXY	PXY	PXY'	PDF
0	1	8	0	2.252	0.932	0.000	0.572	1.202
0	2	8	0	2.224	0.904	0.000	0.544	1.207
0	3	8	5	1.551	0.738	27.692	27.692	0.807
0	4	8	4	1.320	0.440	21.134	21.134	0.446
0	5	8	13	1.302	0.485	75.839	75.839	0.646
1	2	0	0	0.831	0.171	0.000	0.000	0.061
1	3	0	5	1.993	0.673	0.000	0.125	1.365
1	4	0	4	1.928	0.608	0.000	0.065	0.896
1	5	0	13	2.063	0.743	0.000	1.031	1.217
2	3	0	5	1.832	0.512	0.000	0.065	1.286
2	4	0	4	1.949	0.629	0.000	0.070	0.937
2	5	0	13	1.927	0.607	0.000	0.683	1.133
3	4	5	4	1.209	0.476	1.476	1.476	0.582
3	5	5	13	1.384	0.431	97.957	97.957	0.864
4	5	4	13	1.497	0.488	115.444	115.444	0.844

Comparison based on Range. Next we compared the results obtained using the WB and CXY distance functions with PDF. The range of the pair-wise distance computed by the WB distance function is 0.63, that of the CXY distance function is 0.82, and that of PDF is 0.95. The low range of the WB distance function will make it difficult to implement this distance function where polygons need to be grouped based on pair-wise similarity. The CXY distance function which has the same structure PDF improves upon the range of distance values, however, PDF offers the best range of pair-wise distances between polygons among these distance functions. This makes it more suitable for clustering type applications.

Comparison based on Ordering. If we look at the ordering of the pair-wise distances computed using the three distance functions we find that while they all agree on the pair of polygons that are most similar to each other (Polygons 1 and 2 – Rank 1), they do not agree on the pair that are most dissimilar. The complete ranking of pair-wise distances between the polygons is presented in Table 4. In this case, the WB distance function is more bent towards the difference in the non-spatial attributes and as a result Polygons 3 and 4 get assigned a higher rank

(Rank 2 – meaning they are more similar to each other) as compared to the CXY distance function which assigns this pair a lower rank (Rank 4) because the spatial distance between them is greater. On the other hand, as PDF is more rounded, using more spatial and non-spatial attributes it assigns this pair a more appropriate rank (Rank 3) as these polygons are very similar in their non-spatial attributes, and their geographic distance is also not as big as between some other polygons.

Table 4: Ranking of pair-wise distances between polygons

P1	P2	WB Rank	CXY Rank	PDF Rank	WB - CXY	WB - PDF	CXY - PDF
1	2	1	1	1	0	0	0
3	4	2	4	3	2	1	1
0	5	3	5	4	2	1	1
0	4	4	3	2	1	2	1
3	5	5	2	7	3	2	5
4	5	6	6	6	0	0	0
0	3	7	12	5	5	2	7
2	3	8	7	14	1	6	7
2	5	9	8	10	1	1	2
1	4	10	9	8	1	2	1
2	4	11	10	9	1	2	1
1	3	12	11	15	1	3	4
1	5	13	13	13	0	0	0
0	2	14	14	12	0	2	2
0	1	15	15	11	0	4	4
				Sum:	**18**	**28**	**36**
				Average:	**1.2**	**1.87**	**2.4**

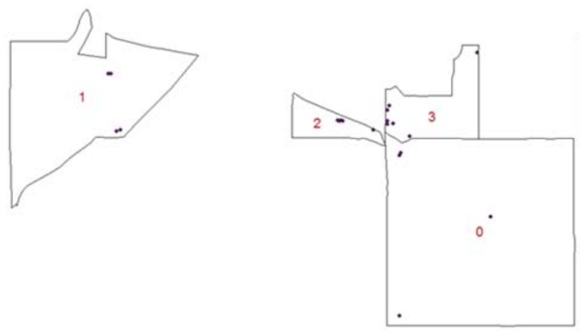
Further, while the WB and CXY distance functions agree on the pair of polygons that are furthest apart (Polygons 0 and 1), PDF assigns this pair a slightly higher rank, and categorizes the pair of polygons 1 and 3 as the furthest apart. This is because the attributes of spatial distribution and spatial density also play an important role. Because of slight clustering within the distribution of liquor licenses in polygon 3, the extrinsic spatial distance between polygons 1 and 3 and polygons 1 and 2 is greater than the extrinsic spatial distance between polygons 0 and 1 and polygons 0 and 2. It is due to this factor that some large shifts in ranking occur within PDF as compared to the WB and CXY distance functions. If we remove the pair of polygons which cause the biggest shifts, we obtain the new ranking shown in Table 5. From Table 5 we can see that the average shift has dropped considerably for both the WB distance function versus PDF, and for the CXY distance function versus PDF. This suggests that we retain the properties of the WB distance function and the CXY distance function for simple polygons, and we add to it, our additional considerations for complex polygons.

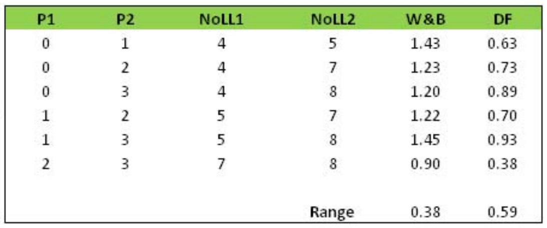
Table 5: Ranking of selected pair-wise distances between polygons

P1	P2	WB Rank	CXY Rank	PDF Rank	WB - CXY	WB - PDF	CXY - PDF
1	2	1	1	1	0	0	0
3	4	2	3	3	1	1	0
0	5	3	4	4	1	1	0
0	4	4	2	2	2	2	0
4	5	5	5	5	0	0	0
2	5	6	6	8	0	2	2
1	4	7	7	6	0	1	1
2	4	8	8	7	0	1	1
1	5	9	9	10	0	1	1
0	2	10	10	9	0	1	1
				Sum:	**4**	**10**	**6**
				Average:	**0.4**	**1**	**0.6**

Comparison based on Correlation. The correlation using the Pearson's correlation function between the WB distance function and the CXY distance function is 0.88, between the WB distance function and PDF is 0.9, and between CXY distance function and PDF is 0.76. A low correlation between PDF and the CXY distance function despite the same structure of the distance functions is once again indicative of the importance of the other spatial attributes of the polygons. A high correlation between PDF and the WB distance function exists for this test dataset because both the distance functions are basically the same in terms of how spatial and non-spatial attributes are combined. The main difference is that distribution & density are used in

PDF, but not in the WB distance function. The distribution and density try to capture variations in spatial objects within the polygons. Due to the normalization factor which is variable, and more localized, within the WB distance function those variations were captured quite well for the sample dataset considered. However, the WB distance function's way of handling local variations is limited. For example, consider the set of polygons (once again a subset of census block groups of city of Lincoln, NE, along with their liquor licenses) shown in Figure 6. When the pair-wise distance was computed for these polygons using the WB distance function and PDF, we found the correlation between these two distance functions dropped significantly and was 0.68.





P1	P2	NoLL1	NoLL2	W&B	DF
0	1	4	5	1.43	0.63
0	2	4	7	1.23	0.73
0	3	4	8	1.20	0.89
1	2	5	7	1.22	0.70
1	3	5	8	1.45	0.93
2	3	7	8	0.90	0.38
			Range	0.38	0.59

Figure 6: Subset Sample Dataset 2, along with the pair-wise distances between the various polygons.

2.4.2 Spatial Clustering Application

Our novel dissimilarity function, PDF, can be seamlessly integrated within any algorithm that uses a distance measure in order to analyze spatial polygons. Here we demonstrate the application of PDF to the *k*-medoids clustering algorithm in order to perform polygon-based spatial clustering. In our clustering, no explicit cluster centre exists. The mean distance between a polygon P_i and all polygons within each cluster determines the membership of P_i. The algorithm terminates when it reaches the maximal number of iterations or the membership of each polygon no longer changes.

We first give two examples that show the benefits of the addition of organizational (extrinsic spatial) attributes in computing the similarity between two polygons. The role of linear object features is illustrated by the following intuitive example where we analyze the clustering results. When the polygons shown in Figure 7(a) are clustered into one cluster, the validity index of the cluster is 76.92. When the linear feature that is shared by all the polygons is added, as shown in Figure 7(b) the validity index increases to 111.11. This increase is directly attributable to the addition of the linear feature which is shared by all the polygons present in the cluster. As there is no explicit center for our clusters, we adapt the existing distance-based validity measure (Turi and Ray 1998). A good clustering result should have a low intra-cluster distance and a high inter-cluster distance. The intra-cluster distance is the average distance between each pair of polygons in a cluster, and the inter-cluster distance is the average distance between each pair of polygons in two clusters. Thus, a high-quality cluster is one whose validity index is large. The clustering result with the maximum validity measure gives us the optimal number of clusters.

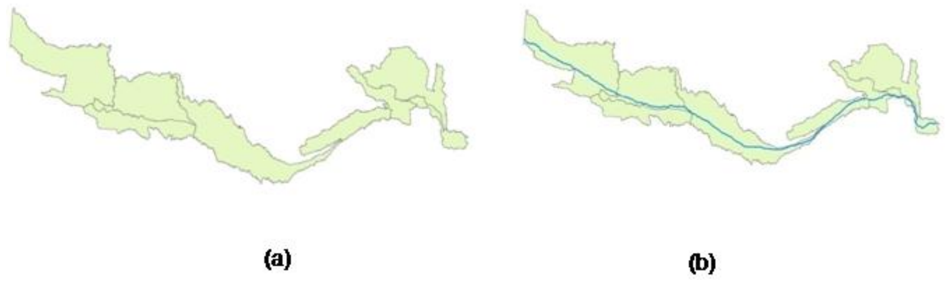


Figure 7: (a) Polygons (subset of watersheds in Nebraska) used to form a cluster (b) Polygons along with linear spatial objects.

Another example shown in Figure 8 illustrates the importance of using areal objects in determining the dissimilarity of polygons. When polygons are grouped into one cluster (Figure 8(a)), the validity index obtained is 83.33. After the addition of the lakes present within each polygon, as shown in Figure 8(b), the validity index rises to 166.66. This increase in the validity index shows that the addition of areal objects makes the clusters more cohesive, i.e. it increases the similarity within the polygons that belong to the same cluster.

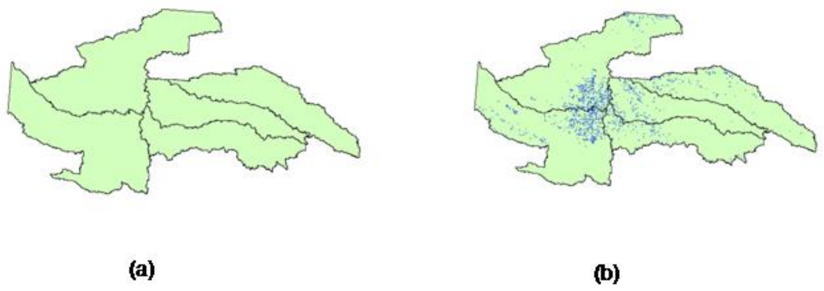


Figure 8: (a) Polygons (subset of watersheds in Nebraska) used to form a cluster (b) Polygons along with areal spatial objects.

2.4.2.1 Watershed Analysis

The dataset comprises of 69 watersheds within the state of Nebraska (Figure 9). A watershed is a geographic region draining into a river, river system, or other body of water. It is a useful areal unit of analysis for many applications including drought and water resource monitoring. The main goal for this set of experiment was to cluster together watersheds that exhibit similar hydrological behavior, and are spatially contiguous. Spatially contiguous clusters of watersheds are useful in many applications. For example, the improvement of the economic efficiency in the reduction of diffused water pollution rests on the identification and formation of homogenous groups of contiguous administrative units of a watershed. By jointly implementing pollution reduction measures, these homogenous groups are able to diminish negative spillover effects and externalities. To identify homogenous groups of administrative units within this watershed cluster analysis methods are used. As the implementation of joint pollution mitigation measures is only sensible and manageable in contiguous areas, the spatial relationship among administrative units is an essential variable for this cluster analysis (Huchtemann & Frondel, 2010).

Data Processing and Feature Selection

Table 6 lists the attributes that are used for clustering watersheds. There are over eight hundred hydrological observation stations including surface water stations, ground water stations, and weather stations. The measurements taken at the various stations – surface water, ground water, and precipitation, cannot be directly used in the clustering process. Therefore, taking the time series data collected from these stations, we found the correlation between watersheds based on their respective surface water, ground water, and weather stations.

Table 6: Attributes for Watersheds

Non-spatial Attributes	Correlation between surface water stations Correlation between ground water stations Correlation between precipitation stations
Intrinsic Spatial Attributes	Set of vertices of the watershed, Elongation of the watershed, Orientation of the watershed
Extrinsic Spatial Attributes	
Point Object Attributes	None
Linear Object Attributes	Major Streams
Areal Object Attributes	Lakes

Figure 9 shows the watersheds in the state of Nebraska along with the various spatial objects used in the process of clustering. The linear objects are the lines (rivers) going across several watersheds, and the areal objects are the polygons (lakes) present within the watersheds.

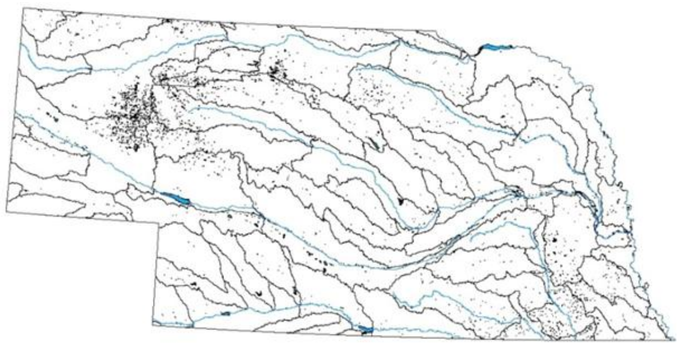


Figure 9: Dataset for the first experiment – Watersheds in the state of Nebraska along with selected streams and lakes used as spatial objects

Clustering Results for the Watershed Dataset

In order to observe the effects of the inclusion of different polygonal attributes in the clustering process, we conducted experiments by using different combinations of the attributes of the polygons/watersheds. When the watersheds are clustered using only their non-spatial attributes, the watersheds with similar correlation indexes are clustered together. Their location in space has no relation with the clustering process. Therefore, we get disjoint clusters in space. In Figure 10, the left side ((a), (c), (e)) shows the clustering result when k = 3 and the right side ((b), (d), (f)) shows the clustering results when $k = 4$. Clusters formed using only the non-spatial attributes (Figure 10(a) & Figure 10(b)) are widely dispersed in space. With the addition of organizational attributes, the spatial organization of the watersheds has its affect in the form of the density of lakes within the watersheds, and the location of the streams – within the watersheds or exterior – changes the similarity of the watersheds, and that in turn results in better quality of clustering (Figure 10(c) & Figure 10(d)). If we compare Figure 10(a) with Figure 10(c) we can see that the distribution and density of lakes have a significant impact on the clustering process. The watersheds with a high density of lakes clustered together belong to the same cluster in Figure 10(c) while they were clustered into different clusters in Figure 10(a) when the organizational attributes were not taken into account. Finally, when we add the structural attributes, watersheds located adjacent to each other in space and sharing a boundary are clustered together (Figure 10(e) & Figure 10(f)) because the spatial structure of the polygons within the geographic space plays an important role in clustering adjacent watersheds together. Therefore, we not only get the clusters with the highest quality, but they are spatially contiguous as well with the addition of the spatial structure and organization of the watersheds along with their correlation indices.

Table 7 lists validity indexes for different combinations of *k* (number of clusters) and different combinations of non-spatial, structural attributes and organizational attributes using the polygonal data for the watersheds. In Table 7, the highest validity index is obtained when $k = 3$ and all the three types of attributes – non-spatial, structural attributes, and organizational attributes, are used for clustering. This suggests that the best quality clusters are formed when the number of clusters is equal to 3, and all the three categories are attributes are taken into account. As for the remaining validity indexes there is no visible pattern, and therefore, no clear conclusions can be made. Note: If the number of polygons within a cluster is less than two, then the validity index will be undefined.

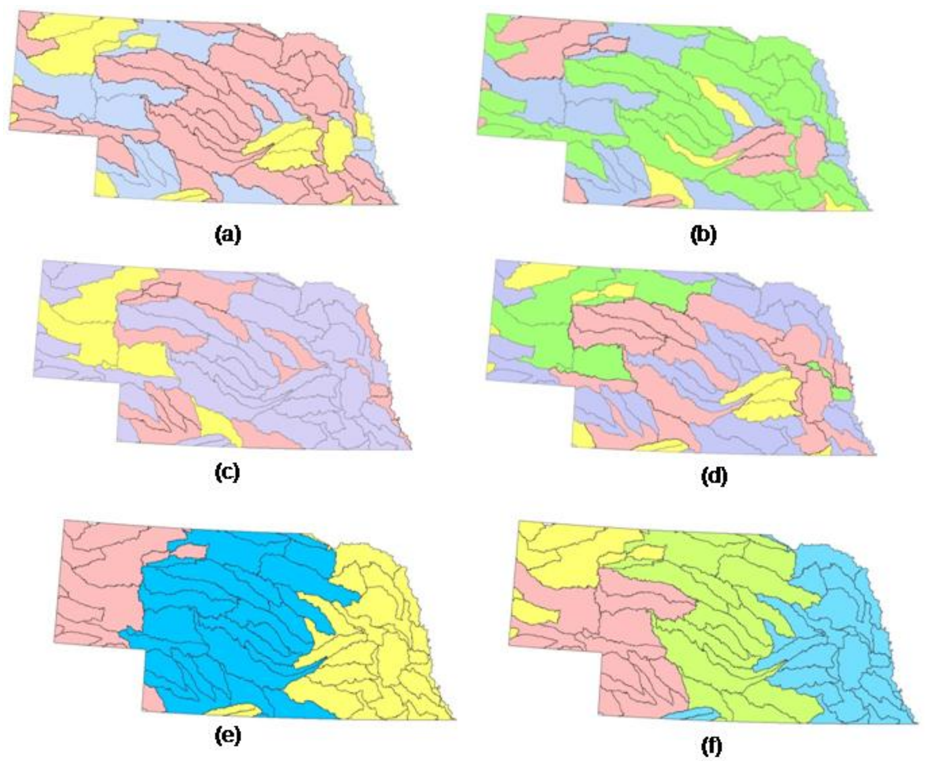


Figure 10: Result of clustering watersheds with. k = 3 ((a),(c),(e)) and k = 4((b),(d),(f)) using different combinations of non-spatial, structural and organizational attributes.

Next, we also test the validity of our clustering results using the gap statistic (Tibshirani et al. 2001) that is used to discover the number of clusters that exist in the dataset. The gap statistic was computed using the gap function defined in the statistical package SAGx written in R. The results for the watershed dataset are shown in Table 8. For the watershed dataset $k = 3$, which matches the result of our validity index.

Table 7: Clustering results for Watershed Dataset

k	**Validity Index, η_{GDF}**			
	Non-Spatial Attributes Only	**Non-Spatial and Extrinsic Spatial Attributes**	**Non-spatial and Intrinsic Spatial Attributes**	**Non-spatial, Intrinsic Spatial and Extrinsic spatial Attributes**
3	76.92	83.33	111.11	125.00
4	0.55	3.80	0.47	8.85
5	35.71	3.34	66.66	1.56
6	0.76	0.43	2.39	1.24

Table 8: Gap Statistic results for the watershed dataset

k	Gap Statistic
2	-0.28
3	-0.23
4	-0.47
5	-0.38
6	-0.31

2.4.2.2 Grouping Counties

For the second experiment we have taken the 93 counties of the state of Nebraska (Figure 11) as the set of polygons. The motivation behind selecting this dataset is the fact that counties have been partitioned into clusters for a long time. There are several applications where counties are divided into groups by the government for jurisdiction purposes, such as congressional districts, natural resource districts, etc. that pertains to non-spatial and spatial attributes. The goal is once again to cluster counties that are similar to each other, and are spatially contiguous. Spatially contiguous clusters of counties are important for applications involving resource distribution and allocation. For example, in resource allocation problems involving redistricting, zones or districts need to be defined where each district is a spatially contiguous cluster of counties or census tracts or some other underlying spatial structure. If one tries to form these districts using a traditional distance function such as Euclidean distance function or the Manhattan distance function, without any additional constraints added to the clustering process, the result would be districts that are spatially disjoint (Joshi et al. 2009c).

Data Processing and Feature Selection

Each county is represented by the set of attributes listed in Table 9. Figure 11 shows the counties along with the organizational attributes used in clustering. For point objects we use the cities (towns). In order to show the effect of linear features, we selected three highways – State route 2, 6, and 20, running across the state.

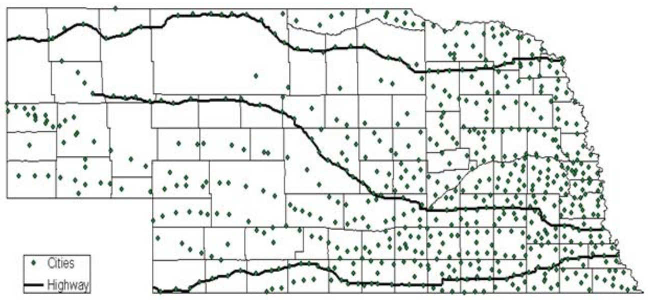


Figure 11: Dataset for the second experiment – Counties in the state of Nebraska along with the point and linear spatial objects

Table 9: Attributes for Counties polygons

Non-spatial Attributes	Total Population
Intrinsic Spatial attributes	Set of vertices of County polygons, Area
Extrinsic Spatial Attributes	
Point Object Attributes	Towns
Linear Object Attributes	Selected Highways
Areal Object Attributes	None

Clustering Results for County Dataset

Several experiments were conducted using different combinations of the attributes of the polygons. Our aim was to see the effect of the structural attributes and organizational attributes in clustering process. In Figure 12, the left side ((a), (c), (e)) shows the clustering result when $k = 3$ and the right side ((b), (d), (f)) shows the clustering results when $k = 4$. It is observed once again that the clusters become more compact and spatially contiguous as we add more spatial features (structural attributes and organizational attributes) to the clustering process. The addition of structural attributes plays a big role in producing contiguous clusters. A closer inspection shows that while clustering using non-spatial attributes only (Figure 12(a)), when $k = 3$, one of the clusters consists of only one county with the largest area. Therefore, area turns out to be the dominating attribute in this clustering process. Using non-spatial attributes only, when $k = 4$, we see that Douglas and Lancaster counties — the counties containing the most populated cities in Nebraska are clustered into one cluster, and no other county is clustered along with them. Cherry county — the county with the largest area also remains in a cluster of its own. This is clearly not the way we would want to cluster counties when we want to partition them for any jurisdiction purpose where we would want a more uniform number of counties (or population) in each cluster.

With the addition of organizational attributes in the clustering process (Figure 12(c)), we see that the result of clustering improves – number of counties in the clusters is more even, and zero singleton clusters are produced. Even though, area still seems to be dominating the clustering of the largest counties together, the point object density and spatial distribution has an even bigger impact as it breaks the previously large clusters into smaller compact clusters. The counties with a larger number of towns are now clustered together. As most of the towns are located along the highways, they have both an indirect and direct impact on the process of clustering. They directly influence clustering since the highways are used as linear spatial object attributes in clustering. They also indirectly influence the process since the locations of many of the towns are guided by highways. Finally, when we add structural attributes to the clustering process, we get clusters that are a lot more uniform (Figure 12(e)) with almost the same number of counties in each cluster, and that are spatially contiguous. The reason behind spatial contiguity is our *boundary adjusted Hausdorff distance* that also takes into account the extent of boundary shared between two polygons. As a result the distance between two polygons sharing a boundary reduces, and we get clusters of polygons located adjacent to each other in space. Table 10 lists validity indexes obtained for different values of *k* with different combinations of the attributes.

As can be seen from Table 10, using non-spatial attributes only produces clusters that have a very low validity index, implying low quality clusters. As we add spatial attributes – structural attributes and organizational attributes, to the non-spatial attributes the validity index increases significantly. When clustering is performed taking into account the non-spatial and organizational attributes, an increase in the validity index is observed. This suggests that more information about the polygons is needed to increase the quality of clustering. When clustering is performed using non-spatial and structural attributes, the validity index increases further and we do not get any more bad clusters (with validity index of infinity). This implies that good quality clusters are formed using these two sets of attributes. However, when clustering was performed using all three attribute sets – non-spatial, structural attributes, and organizational attributes, we found that the validity index increased even further, and the largest validity index ever was produced for k =4. This suggests that the best quality clusters are formed using all the three types of attributes together. To further evaluate our clustering results we have applied the gap statistic to the county dataset to discover the number of clusters that exist in the dataset. The results of gap statistic are shown in Table 11. For the county dataset $k = 4$, which once again matches the result of our validity index.

2.4.2.3 Summary

Based on our experimental analysis on the clustering application, we have observed that with the addition of the spatial attributes – both the structural and organizational attributes – within the dissimilarity function allows for a more accurate comparison of the polygons, and helps us achieve our goal of forming spatially contiguous and compact clusters using the *k*-medoids clustering algorithm. For example, when clustering is based upon non-spatial attributes alone the result is spatially disjoint clusters (Figures 10(a), 10(b), 12(a), 12(b)). Therefore, the principles of spatial autocorrelation and spatial heterogeneity are not met. The addition of spatial attributes produces clusters that are spatially contiguous and compact (Figures 10(e), 10(f), 12(e), 12(f)). Thus we have shown that even though *k*-medoids is not a favored algorithm in this domain, with the use of the appropriate distance/dissimilarity function, the desired results can be produced using any distance-based algorithm such as *k*-medoids.

Furthermore, we have demonstrated the robustness of our dissimilarity function by applying it on two real and yet spatially different datasets – the watershed dataset, and the county dataset. The watershed dataset had spatial attributes resulting primarily from natural factors. The county dataset, on the other hand, is derived from human decisions. Due to the underlying simila-

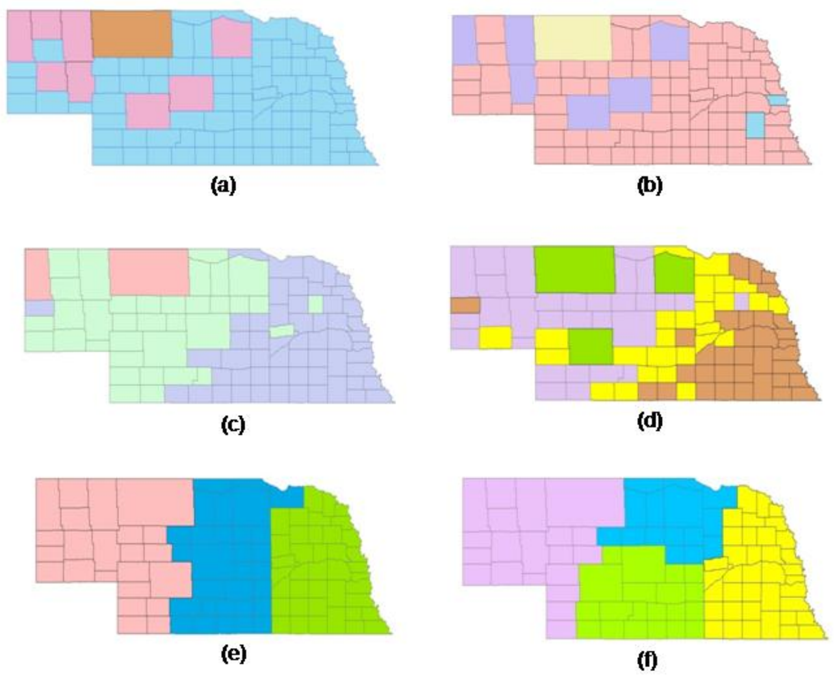


Figure 12: Result of clustering counties with. $k = 3$ and $k = 4$ using different combinations of non-spatial, structural and organizational attributes.

Table 10: Clustering results for County Dataset

k	Validity Index, η_{GDF}			
	Non-Spatial Attributes Only	Non-Spatial and Extrinsic Spatial Attributes	Non-spatial and Intrinsic Spatial Attributes	Non-spatial, Intrinsic Spatial and Extrinsic spatial Attributes
2	0.004	8.55	18.52	18.18
3	Undefined†	29.41	21.28	19.23
4	Undefined†	0.61	7.35	22.72
5	Undefined†	0.43	4.72	20.00
6	Undefined†	Undefined†	16.39	16.39

†The validity index has a value of undefined when the number of polygons within a cluster is one.

Table 11: Gap Statistic results for the county dataset

k	Gap Statistic
3	0.13
4	0.16
5	0.13
6	0.26

-rity within the behavior of every watershed, there do not exist any well-separated clusters. As a result all the Gap values for the watershed dataset in Table 8 are below zero. On the other hand, for the county dataset, since each county has distinct dissimilarities with respect to number of towns and population, there exist well-separated clusters. As a result all the gap values in Table 11 are above zero. However, as can be observed from Tables 7 and 10, using our adapted validity index, we have clear notion on the improvement of the clustering quality with the addition of the spatial attributes. For example, in the watershed dataset, when $k = 3$, and clustering is performed using only the non-spatial attributes, the validity index is 76.92. With the addition of the organizational attributes to the non-spatial attributes, the validity index now increases to 83.33. When, on the other hand, the structural attributes were added to the non-spatial attributes, the validity index increases by a bigger margin and is now at 111.11. Further improvement is observed in the validity index of the clusters when all three types of attributes were used within the dissimilarity function, and the index has now increased to 125.00. This result matches with the visual inspection of the clusters, and the clusters become more spatially contiguous and compact in Figure 10(e). A similar result is also observed in the county dataset experiment.

2.5 Conclusions and Future Work

While dissimilarity functions play a central role in many applications/domains, such functions in the context of geospatial polygons are not well studied. In such domains, ignoring the spatial aspects in dissimilarity is neither correct nor does it lead to accurate results. In this research, we have developed a new similarity/dissimilarity measure for polygons known as the Polygonal Dissimilarity Function (PDF) that integrates the non-spatial attributes of the polygon with the spatial attributes encoded in the form of the structural and organizational information. In order to incorporate these properties, the polygons are represented using three sets of attributes: *non-spatial* attributes, *intrinsic spatial attributes* and *extrinsic spatial attributes*. In the process of defining our dissimilarity function, we have not only addressed the unary properties of the polygons, but also their binary properties, by taking into account the linear and areal features that may be shared by multiple polygons.

We have shown that our proposed PDF outperforms the other distance functions proposed in literature. It reduces the distance between the polygons that are most similar, and increases the distance between the polygons that are most dissimilar. It takes a balanced yet flexible approach to incorporate the influence of spatial and non-spatial aspects. Our comparative analysis demonstrates that we retain the properties of the previously used well established distance functions such as the WB distance function and the CXY distance function for simple polygons yet is able to include additional considerations for complex polygons.

In essence, we have proposed a framework to include all the attributes of the polygons in order to measure the similarity/dissimilarity between polygons. This framework will allow us to administer a systematic approach to experimenting with different combinations of attributes by giving the freedom to the user to change the weights of the different parts of the dissimilarity function. Further, by taking into account computational complexity, with such a framework, whether and how to develop an automated system for carrying out the clustering task or identifying the appropriate attributes for clustering can be informed. Moreover, domain knowledge and human expertise can be factored into the framework, for example, to determine the relative weights of the various attributes. Thus, with this framework, we also aim to lend more structure to the search process for the correct combination, inform algorithm developers of computational complexity for automation, and better represent and engineer domain knowledge and human expertise. Our novel dissimilarity function can thus be seamlessly integrated into any polygon analysis algorithm that uses a distance function without added computational complexity.

While we have developed a dissimilarity function that takes the spatial analysis of polygons a step further, there are many open questions that have not been addressed by our work. Additional distance measures in the object space as well as more complex form of the combination function that integrates the distances in the three spaces can be explored. One could design a distance measure for linear objects using their density and a distance measure for areal objects that takes into account the amount of sharing, and topology. Our approach can be extended to fuzzy algorithms that will also have use in many geospatial applications. Another direction is to include nominal, categorical, and ordinal attributes in our dissimilarity function to simplify our methodology. We plan to investigate how to flexibly map the differences in values for these different attributes onto a comparable scale.

Acknowledgements

We would like thank to David Marx for his valuable insight and feedback. We would also like to extend our thanks to Lei Fu, Bill Waltman, and Tao Hong for their assistance in data processing and preparation for this research.

Publications

This chapter appears in the following:

1. Joshi, D., Samal, A., & Soh, L-. K. (2009). A Dissimilarity Function for Clustering Geospatial Polygons. *17th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009)*, (pp. 384-387). Seattle, WA.
2. Joshi, D., Samal, A., & Soh, L-. K. (under review). A Dissimilarity Function for Polygons, submitted to *Journal of Geographic Systems* in Decemeber 2010.

Chapter 3: Density-Based Clustering of Polygons

3.1 Introduction

Clustering is the process of unsupervised classification that is fundamental to spatial data mining and spatial analysis. Several spatial clustering algorithms have been proposed in the past (see Section 3.2.1). However, most of them are focused on clustering point data sets. There are several applications of spatial clustering where clustering algorithms for point datasets may not give efficient results. This mainly happens when polygons need to be clustered instead of points. For example, an important application of polygonal clustering is the process of regionalization. Regionalization is the process of region building where smaller units (polygons) are grouped together into larger contiguous regions based on some attribute or criteria. Thus, regionalization produces clusters of polygons that are spatially compact and contiguous. If polygons are indeed represented as points and clustering is performed, the spatial information and relationships between polygons are not captured and utilized during the clustering process. Due to the inadequacies of the point-based clustering algorithms new clustering algorithms need to be developed in order to cluster polygons. In this chapter we propose a novel algorithm P-DBSCAN for clustering polygonal datasets.

Our algorithm P-DBSCAN is based on the well established density-based clustering algorithm DBSCAN (Ester, Kriegel, Sander, & Xu, 1996). There are several advantages of using DBSCAN as our reference algorithm. First, it has the ability to discover clusters of arbitrary shapes such as linear, concave, and oval. Second, DBSCAN does not require the number of clusters to be determined in advance. Finally, DBSCAN is scalable to be used with large databases. The new algorithm P-DBSCAN extends DBSCAN to cluster *polygons* instead of points by redefining the concepts of the neighborhood of a polygon, core polygon, border polygon, and noise polygon. The clustering is done based on the distance between two polygons leading to the polygons close to each other being clustered together, and thus resulting in spatially compact clusters.

Note that a key component of our P-DBSCAN algorithm is the calculation of the distance function (see Section 3.3.2). Using this distance function, both contiguous polygons and disjoint polygons can be clustered using our novel algorithm. When the polygons are contiguous in space, the extent of the boundary shared by two polygons is taken into account while computing the distance between them. On the other hand, if the polygons are disjoint, the shared boundary component is ignored. PDBSCAN is not restricted to polygons in 2-D space only, and is applicable to polygons in n-dimensional space, with $n > 2$.

The rest of the chapter is organized as follows. Section 3.2 presents the related work giving a background on spatial clustering and density-based spatial clustering. Section 3.3 defines the density-based concepts for polygons, our methodology for computing the distance between two polygons, and explains our algorithm in detail. Section 3.4 presents an application of our clustering algorithm. Finally, our conclusion and directions for future work are given in Section 3.5.

3.2 Related Work

In this section we present a background on different spatial clustering algorithms. Following which, we present an overview of the density-based clustering concepts for points.

3.2.1 Spatial Clustering Algorithms

Clustering algorithms can be categorized into five main types: Partitional, Hierarchical, Density-based, Grid-based, and Model-based clustering algorithms. In Partitional algorithms, partitions of a database D are developed, and a set of clusters are formed. The number of clusters generated has to be specified in advance. The cluster similarity is measured with respect to the mean value (cluster center) of the objects in a cluster. Examples are PAM (Ng & Han, 1994), CLARA (Ng & Han, 1994), and CLARANS (Ng & Han, 2002).

Hierarchical algorithms create a hierarchical decomposition of the database. This hierarchical decomposition is represented as a dendrogram. Each level of the dendrogram represents a set of clusters. Thus, a set of nested clusters organized as a hierarchical tree are produced. As a result the initial knowledge of the number of clusters is no longer required. However, a termination condition needs to be specified. Examples of hierarchical clustering are CURE (Guha, Rastogi, & Shim, 1998) and BIRCH (Zhang, Ramakrishnan, & Linvy, 1996).

Density-based clustering algorithms are based on the idea that objects which form a dense region should be grouped together into one cluster. These algorithms search for regions of high density in a feature space that are separated by regions of lower density. Thus, density-based methods can be used to filter out noise, and discover clusters of arbitrary shape. Examples of density-based clustering algorithms are DBSCAN (Ester, Kriegel, Sander, & Xu, 1996), DENCLUE (Hinneburg & Keim, 1998), and OPTICS (Ankerst, Breunig, Kriegel, & Sander, 1999).

Grid-based algorithms are based on multiple level grid structure. The entire space is quantized into a finite number of cells on which operations for clustering are performed. Summarized information about the area covered by each cell is stored as an attribute of the cell. The main advantage of this approach is its fast processing time. However, the summarized information leads to loss of information. Examples of grid-based clustering algorithms are STING (Wang, Yang, & Muntz, 1997), WaveCluster (Sheikholeslami, Chatterjee, & Zhang, 1998), and CLIQUE (Agrawal, Gehrke, Gunopulos, & Raghavan, 1998).

In model-based algorithms a model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each cluster. They are often based on the assumption that the data are generated by a mixture of underlying probability distributions. COB-WEB (Fisher, 1987) is an example of this approach.

We select the density-based approach for clustering polygons since there is no need to know the number of clusters in advance as required in partitional algorithms, nor is there a need to store summarized information as in grid-based algorithms. Moreover, polygons in geographic space and in many other domains naturally respond to the density-based approach. For example, in geographic space, we have a set of contiguous polygons, and another set of polygons located far away from the first set. At a larger scale, these two sets will belong to a cluster each, thus corresponding to clusters formed where the object density is high.

3.2.2 Density-Based Concepts for Points

A density-based clustering algorithm hinges upon the assumption that a valid cluster must have sufficient density. Ester et al. proposed a density-based clustering algorithm used for clustering point datasets, called DBSCAN (Ester, Kriegel, Sander, & Xu, 1996). Here we list the main concepts of density for points as defined in (Ester, Kriegel, Sander, & Xu, 1996). These concepts are later (see Section 3.3.1) extended in our clustering algorithm P-DBSCAN for clustering polygons.

Definition 1: (ε-neighborhood of a point) The ε-neighborhood of a point p, denoted by $N_\varepsilon(p)$, is defined by $N_\varepsilon(p) = \{q \in D | dist(p, q) \le \varepsilon\}$.

Definition 2: (directly density-reachable) A point pp is *directly density-reachable* from a point qq wrt. $\varepsilon, MinPts$ if 1) $p \in N_\varepsilon(q)$ and 2) $|N_\varepsilon(q)| \ge MinPts$ (core point condition).

Directly density-reachable is symmetric for pairs of core points. In general, however, it is not symmetric if one core point and one border point are involved.

Definition 3: (density-reachable) A point p is *density reachable* from a point q wrt. ε, $MinPts$ if there is a chain of points $p_1, \ldots, p_n, p_1 = q, p_2 = p$ such that p_{i+1} is directly density-reachable from p_i.

Definition 4: (density-connected) A point pp is *density connected* to a point qq wrt. ε, and if there is a point o such that both, p and q are density-reachable from o wrt. ε, $MinPts$. Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive.

Definition 5: (cluster) Let D be a database of points. A *cluster* C wrt. $\varepsilon, MinPts$ is a non-empty subset of D satisfying the following conditions:

1) $\forall\, p, q$: if $p \in C$ and q is density-reachable from p wrt. ε and $MinPts$, then $q \in C$. (Maximality)
2) $\forall p, q \in C$: p is density-connected to q wrt. ε and $MinPts$. (Connectivity)

Definition 6: (noise) Let $C_1, \ldots, C_k$ be the clusters of the database D wrt. parameters ε and $MinPts$, then we define the *noise* as the set of points in the database D not belonging to any cluster C_i, i.e. $noise = \{p \in D | \forall i: p \notin C_i\}$.

3.3 Density-Based Clustering of Polygons

3.3.1 Density-Based Concepts for Polygons

Since polygons are spread out in space, factors that would have no effect on points—such as topology and direction—come into play. Also, if the polygons are share boundaries, then two polygons sharing a larger extent of their boundary should be considered closer to each other as compared to two polygons sharing a very small portion of their boundaries. This conclusion follows from the observation that more close two polygons are to each other, more similar they will be in their characteristics. As a result of these factors, some of the density-based concepts for points do not directly apply to polygons. Mainly, the concept of a *core polygon* and its neighborhood are fundamentally different from that of a core point. Once a core polygon is defined, and the polygons that belong to its neighborhood, the same concepts of directly-density reachable, density-reachable, and density-connected for points can then be applied to polygons. In the following, we formalize the density-based concepts for polygons.

ε-neighborhood of a Polygon: The *ε-neighborhood of a polygon* p, denoted by $N_\varepsilon(p)$, is defined by $N_\varepsilon(p) = \{q \in D | dist(p, q) \le \varepsilon\}$, where D is the data set of polygons, and $dist(p, q)$

is defined as the distance between polygons p and q. For example in Figure 1, the ε-*neighborhood of the polygon* p is $N_\varepsilon(p) = \{a, b, c, d, e, f, g\}$.

Radial Spatial Neighborhood of a Polygon: The neighborhood of a polygon can be further partitioned. That is, $N_\varepsilon(p) = \bigcup_{i=1}^{R} N_{\varepsilon,i}(p)$ such that R is the number of equal-size sectors radially partitioning the space around the polygon p. The definition of $N_{\varepsilon,i}(p)$ extends directly from the ε-neighborhood of the polygon, but only looks at the sector indexed by i. Figure 13 shows an example of the radial spatial neighborhood of a polygon p (shaded).

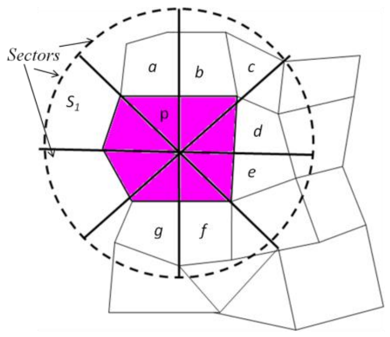


Figure 13: $\mathbf{R = 8}$ Radial spatial partitions of a polygon's neighborhood. Note that here the first sector is $\mathbf{S_1}$ as shown, and the ordering is clockwise. This is arbitrary for illustration purpose.

The radial spatial neighborhood of polygon p in Figure 13 is divided into 8 sectors: $S_{1,}, \ldots, S_8$. Therefore, $N_{\varepsilon,1}(p) = \{b, c\}$, $N_{\varepsilon,2}(p) = \{c, d\}$, $N_{\varepsilon,3}(p) = \{d, e\}$, $N_{\varepsilon,4}(p) = \{e, f\}$, $N_{\varepsilon,5}(p) = \{g\}$, $N_{\varepsilon,6}(p) = \emptyset$, $N_{\varepsilon,7}(p) = \emptyset$, $N_{\varepsilon,8}(p) = \{8\}$. Thus, $\bigcup_{i=1}^{R} N_{\varepsilon,i}(p) = \{a, b, c, d, e, f, g\}$ which is the same as $N_\varepsilon(p)$.

Core Polygon: A *core polygon c* is defined as a polygon that has at least a minimum number of polygons (*MinPolys)* within its ε-neighborhood, and there are at least a minimum number of radial spatial partitions (*MinS)* that are non-empty, i.e. $\left[Count_{i=1}^{R}\left(N_{\varepsilon,i}(c) \neq \emptyset\right)\right] \geq MinS$. For example, in Figure 14, if ε *=1, MinPolys = 4 and MinS = 8, p, o* and q are core polygons.

Border Polygon: A *border polygon b* is defined as a polygon that has more than $R - MinS$ of its radial spatial partitions empty, i.e. $\left[Count_{i=1}^{R}\left(N_{\varepsilon,i}(b) = \emptyset\right)\right] > R - MinS$. , where R is the total number of partitions. For example, in Figure 14 with ε *=1, MinPolys = 4 and R = 8, and MinS = 8, b* is a border polygon, since $R - MinS = 0$.

Outlier Polygon: An *outlier polygon* is defined as a polygon that does not have any polygons within the threshold distance of ε.

Directly Density-Reachable: A polygon p is *directly density-reachable* from a polygon q wrt ε, if

1) $p \in N_{\varepsilon}(q)$ *and*

2) q is a core polygon.

Directly density-reachable is symmetric for pairs of core polygons. In general, however, it is not symmetric if one core polygon and one border polygon are involved. For example, in Figure 14 polygon a is *directly density-reachable* from a polygon p, however polygon p is not *directly density-reachable* from a polygon a.

Density-Reachable: A polygon p is *density-reachable* from a polygon q if there is a chain of polygons $p_1, \ldots, p_n$ $where$ $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from $p_i$where $\{i = 1 \ to \ n - 1\}$. In Figure 14 polygons p is density-reachable from polygon q.

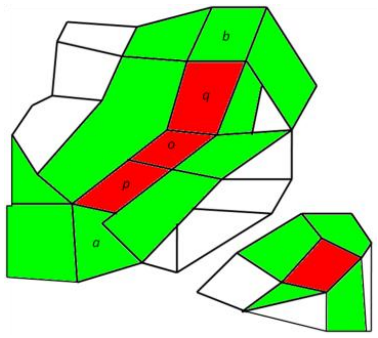


Figure 14: Synthetic set of polygons (Red – Core Polygon, Green - **ε**-neighborhood of the core polygons)

Density-Connected: A polygon p is *density connected* to a polygon q if there is a polygon o such that both, p and q are density-reachable from o. In Figure 14, polygon a and polygon b are density-reachable from polygon o, and thus are density-connected to each other.

Cluster: A *cluster* C wrt. ε is a non-empty subset of D satisfying the following conditions:

1) *Maximality*: $\forall p, q | p \in C, q \in D$ q is density-reachable from p, then $q \in C$.
2) *Connectivity*: $\forall p, q \in C: p$ is density-connected to q.

3.3.2 Distance Function for Polygons

Each polygon is represented as a set of vertices that form the boundary of the polygon. We use the Hausdorff distance as the basis for computing the distance between two polygons in the boundary space. The *Hausdorff distance* between two sets of points (Rote, 1991) is defined as the maximum distance of points in one set to the nearest point in the other set. Formally, the Hausdorff distance (D_h) from set A to set B is defined as

$$D_h(A, B) = \max_{a \epsilon A}(\min_{b \in B} d(a, b)) \quad (1)$$

where a and b are points of sets A and B, respectively, and $d(a, b)$ is any distance metric between the two points a and b. The distance metric used within Hausdorff distance in order to calculate the distance between two points is the Euclidian distance.

If the boundaries of the polygons p and q are represented by two sets of points A and B respectively, we use the following defined distance measure (d_h) between two polygons

$$d_h(p, q) = \max(D_h(A, B), D_h(B, A)) \quad (2)$$

Intuitively, we expect the distance between two polygons with shared boundary to be less. However, the standard Hausdorff distance is defined on the set of points and does not incorporate any sharing of the boundary. In order to incorporate this, we define a new distance measure, called the *boundary adjusted Hausdorff distance*, that is inversely proportional to the length of the shared boundary between the two polygons, between two polygons p and q as follows:

$$d_{hs}(p,q) = \left(1 - \frac{2S_{pq}}{S_p + S_q}\right) \times d_h(p,q) \qquad (3)$$

where d_h is the original standard Hausdorff distance, S_p and S_q are the perimeter lengths of polygons p and q, respectively, and S_{pq} is the length of their shared boundary. This distance, d_{hs}, is smaller than the standard Hausdorff distance when two polygons have shared boundary, and becomes the standard Hausdorff distance when two polygons have no shared boundary, i.e., when S_{pq} = 0. We use twice the shared distance in the definition to balance the effect of the denominator.

3.3.3 P-DBSCAN Algorithm

Our algorithm works similar to DBSCAN where we select a polygon p from the dataset D and check if it has been assigned to a cluster already. If the polygon is still unclassified, then the ExpandCluster routine is called. As in DBSCAN, *ExpandCluster* is the where the cluster assignment is done. P-DBSCAN checks whether a polygon is a core polygon or not by calling the *Expandable* method. This method generalizes the method of checking for the coreness of a polygon or any other object being clustered, as opposed to DBSCAN that implicitly checks only for the MinPts condition. If a polygon is classified as a core polygon, its neighbors are retrieved from the database and assigned to the same cluster as the core polygon. Figure 15 presents our proposed P-DBSCAN Algorithm.

DBSCAN now becomes a special case of P-DBSCAN. The time complexity of our algorithm remains the same as DBSCAN that is $nlog(n)$ where n is the size of the database.

3.4 Experimental Analysis

To show the effectiveness of our algorithm we have conducted several experiments and compared our results with DBSCAN. The input to the P-DBSCAN algorithm are the polygons, a pre-

```
P-DBSCAN
Input: D, $\varepsilon$, MinPolys
Output: Set of Clusters
Initially all polygons are UNCLASSIFIED
ClusterId is initialized
For each polygon p in D
    If its ClusterId is UNCLASSIFIED then
        call ExpandCluster.
    If ExpandCluster returns True then
        increment ClusterId
    End If
    End If
End For
```

```
ExpandCluster
Input: p, ClusterId
Output: True or False
If p is Expandable then
    Set the ClusterID of p to ClusterId
    For each neighbor of p,
      Call the ExpandCluster routine.
      Return True.
Else return False.
```

```
Expandable
Input: p
Output: True or False
If p is surrounded by polygons in at least MinS radial spatial partitions then
    Get the $\varepsilon$-Neighborhood of p.
    If $\varepsilon$-Neighborhood of p contains MinPolys polygons then
        Return True
Else return False.
```

Figure 15: P-DBSCAN Algorithm

defined ε and a pre-defined $MinPolys$. R is set to 4, and $MinS$ is set to 4 for all experiments as well. The input to the DBSCAN algorithm are the centroids of the polygons, a pre-defined ε and a pre-defined $MinPts$. To demonstrate the robustness of our algorithm we use two different experiments. We first use a synthetic dataset which is a 10×10 grid of 1×1 unit squares. We then use two real datasets from a practical application, i.e. the census tracts of two states in USA – Nebraska and South Dakota. When DBSCAN was applied on these datasets, the Euclidean distance was computed between the centroids of the polygons in order to measure how close they are to each other. P-DBSCAN uses the modified Hausdorff distance function as described in 3.3.2. All the three datasets are sets of contiguous polygons. Thus, both the algorithms DBSCAN and P-DBSCAN when applied with the appropriate input parameters should result in a single cluster consisting of all the polygons.

3.4.1 Analysis using Synthetic Dataset

The first set of experiments were conducted using a 10 × 10 grid resulting in a dataset with 100 polygons all of the same size and shape. The reason to use this dataset was to show that P-DBSCAN produces the same results as DBSCAN when all the polygons are equidistant from each other, making DBSCAN a special case of P-DBSCAN. In the first test, we applied $\varepsilon = 0.5$ which resulted in zero clusters (Figure 16) since the distance was too small to include any other polygon in its neighborhood. When $\varepsilon = 1.5$ (Figure 17(a)), all the polygons were grouped together in the same cluster by both the algorithms, i.e. DBSCAN and P-DBSCAN.

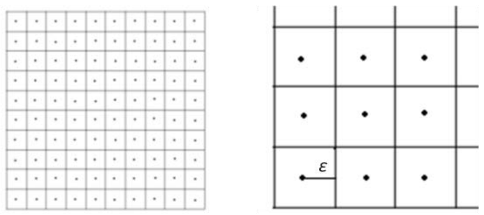


Figure 16: Result of clustering using DBSCAN (a) Polygons used for clustering (b) Expanded version of dataset showing $\boldsymbol{\varepsilon = 0.5}$

Figure 17 shows how the cluster grows upon the application of the DBSCAN algorithm to the dataset. Figure 17(b) shows the first core polygon in red. The surrounding polygons shown in green belong to the ε-neighborhood of the core polygon. Figure 17(c) shows the next core polygon detected. Finally Figure 17(e) shows the entire cluster. All the polygons except the four corner polygons shown in green were marked as core polygons by the algorithm.

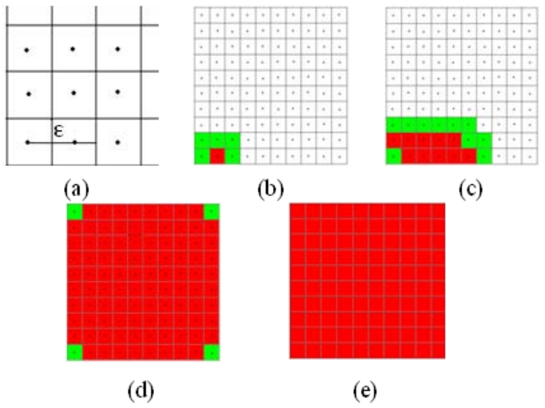


Figure 17: Result of clustering using DBSCAN (a) $\boldsymbol{\varepsilon = 1.5, MinPts = 5}$ (b) First core polygon(Red) and its $\boldsymbol{\varepsilon}$-neighborhood (Green) (c) Consecutive core polygon detected and its $\boldsymbol{\varepsilon}$-neighborhood (d) Further progression of core polygon detection belonging to the same cluster (e) Final result – All polygons belong to the same cluster.

We examine the performance of P-DBSCAN using the same dataset. The spatial neighborhood of a core polygon is divided into $R = 4$ radial partitions with $MinS = 4,$ and $MinPolys = 5$.The result of clustering the polygons shown in Figure 17(a) can be seen in Figure 18.

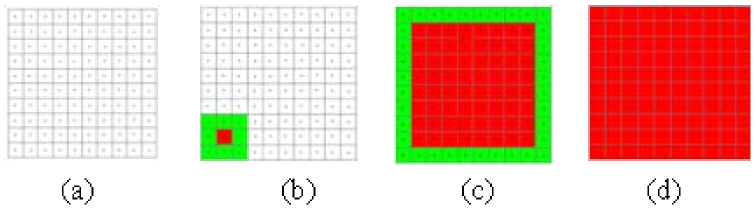


Figure 18: Result of clustering using P-DBSCAN (a) Polygons used for clustering $\boldsymbol{\varepsilon = 1.5, MinPolys = 5, MinS = 4}$ (b) First core polygon(Red) and its $\boldsymbol{\varepsilon}$-neighborhood (Green) (c) Further progression of core polygon detection belonging to the same cluster (d) Final result – All polygons belong to the same cluster

We can see in the above figures that while the core points and core polygons are not the same, both the algorithms resulted in the same cluster consisting of all the polygons in the grid.

3.4.2 Analysis using Real Datasets

Experiments were conducted on two sets of real data - the Nebraska census tract dataset, and South Dakota census tract dataset. The Nebraska dataset (Figure 19) consists of a set of 505 contiguous polygons. Both the algorithms DBSCAN and P-DBSCAN were applied to this dataset using different values of ε, $MinPts$, and $MinPolys$.

The results for DBSCAN with different values of ε and $MinPts$ can be seen in Figure 20. We start with the value ε as average distance between the centroids of the polygons in the dataset which is *0.75*, and $MinPts = 2$ (Figure 20(a)). We find that all the polygons are clustered together to form one large cluster. When $MinPts$ was increased to 5 (Figure 20(b)), the number of clusters did not increase and some polygons were left out from the cluster. With a smaller value of ε more clusters are produced with a large number of polygons being left unclustered.

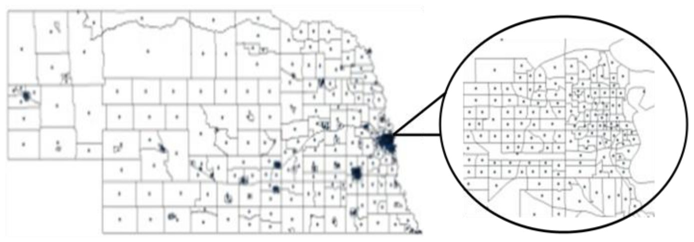


Figure 19: Census Tract Polygons in Nebraska dataset

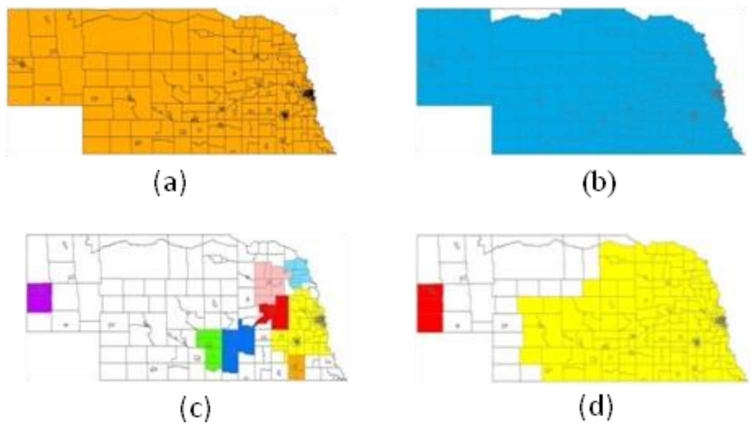


Figure 20: Results of clustering using DBSCAN (a) $\boldsymbol{\varepsilon = 0.75, Minpts = 2}$ (b) $\boldsymbol{\varepsilon = 0.75, MinPts = 5}$ (c) $\boldsymbol{\varepsilon = 0.25, MinPts = 5}$ (d) $\boldsymbol{\varepsilon = 0.5, MinPts = 5}$

The results for P-DBSCAN with different ε value and $MinPolys$ can be seen in Figure 21. Here too we start with $\varepsilon = 0.75$. $MinPolys$ was set to 1, 2, and 5. With $\varepsilon = 0.75, MinPolys = 1$ (Figure 21(a)), it was seen that all the polygons belonged to a cluster leaving no polygons unclustered. As $MinPolys$ was increased (Figure 21(b) & 21(c)), the number of polygons left unclustered increased. When ε was increased, number of polygons belonging to a cluster reduced even further, leaving a lot white space or unclustered polygons within the dataset (Figure 21(d)).On the other hand, when ε was increased, and reached to a value of 2, all the polygons were clustered together to belong to the same cluster. The same trend of number of clusters detected with increasing ε was seen here as well, with all the polygons belonging to the same cluster when $\varepsilon = 2.0$ (Figure 21(f)).

In order to compare the results of both the algorithms shown above, we compute the *compactness of a cluster* using the *Schwartzberg Index* (Schwartzberg, 1996). It measures the compactness of a cluster as the square of the perimeter of the cluster divided by the area of the cluster. The lower the value of this index, the more compact the cluster is.

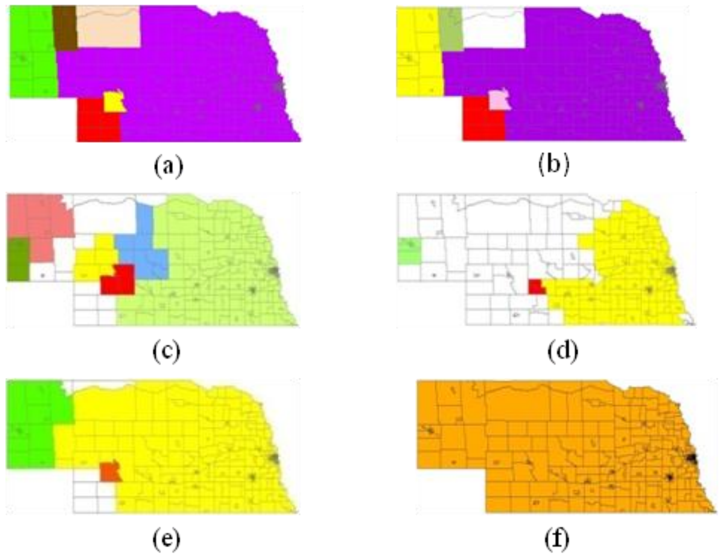


Figure 21: Results of clustering using P-DBSCAN (a) $\boldsymbol{\varepsilon = 0.75, MinPolys = 1}$ (b) $\boldsymbol{\varepsilon = 0.75, MinPolys = 2}$ (c) $\boldsymbol{\varepsilon = 0.75, MinPolys = 5}$ (d) $\boldsymbol{\varepsilon = 0.5, MinPoly = 5}$ (e) $\boldsymbol{\varepsilon = 1.0, MinPolys = 5}$ (f) $\boldsymbol{\varepsilon = 2.0, MinPolys = 5}$

The compactness index was computed for the clusters formed by DBSCAN and P-DBSCAN for the Nebraska dataset. In order to compare the compactness of the clusters formed by both the algorithms, we computed the average all the clusters formed at a given ε and $MinPolys = 5$.The results are shown in Figure 22.

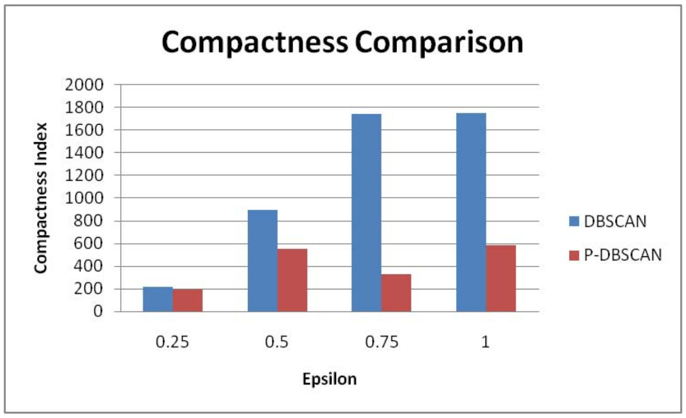


Figure 22: Compactness Ratio for clusters formed using DBSCAN and P-DBSCAN

As shown in Figure 22, P-DBSCAN produces clusters with a lower compactness index. This implies that the clusters formed using P-DBSCAN are spatially more compact than the clusters formed using DBSCAN.

The South Dakota dataset (Figure 23) consists of 236 contiguous polygons. Both the algorithms DBSCAN and P-DBSCAN were applied to this dataset using different values of ε, $MinPts$, and $MinPolys$.

The results of clustering using DBSCAN with different values of ε and $MinPts$ are shown in Figure 24. As before, we start with $\varepsilon = 0.65$ which is the average distance between the polygons in the dataset. As $MinPts$ increases, there are polygons which are left unclustered (Figure 24(b) & 24(c)). As ε is increased to 0.75, all the polygons are clustered together in one polygon. At this point, the value of $MinPts$ has no effect on the clustering process.

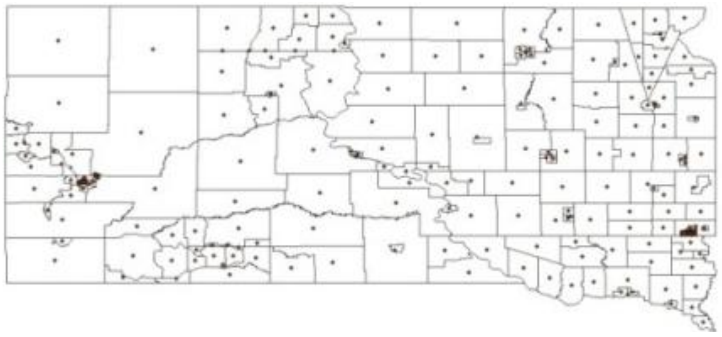


Figure 23: Census Tract Polygons in South Dakota dataset

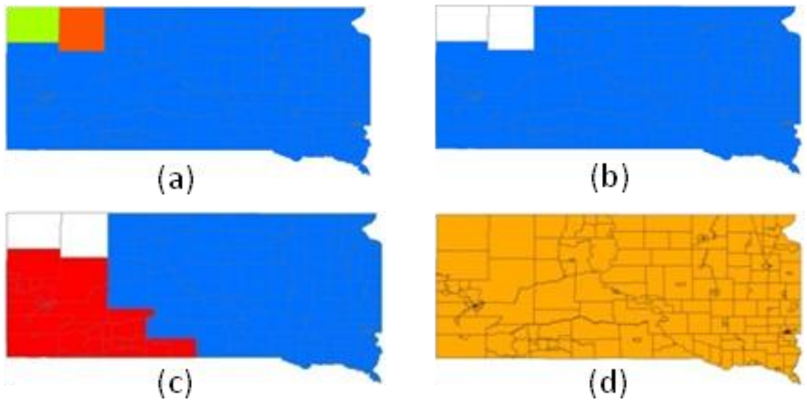


Figure 24: Result of clustering using DBSCAN (a) $\boldsymbol{\varepsilon = 0.65, MinPts = 1}$ (b) $\boldsymbol{\varepsilon = 0.65, MinPts = 2}$ (c) $\boldsymbol{\varepsilon = 0.65, MinPts = 5}$ (d) $\boldsymbol{\varepsilon = 0.75, MinPts = 5}$

The results of clustering using P-DBSCAN with different values of ε and $MinPolys$, and $MinS = 4$ can be seen in Figure 25. As done for DBSCAN, we start the results with $\varepsilon = 0.65$ and $MinPolys = 1$ (Figure 25(a)) we see that all polygons are clustered with none of the polygons left unclustered. The number of clusters is more than DBSCAN, and none of the clusters contains only one polygon. When the value of $MinPolys$ is increased some of the polygons remain un-clustered (Figure 25(b) & 25(c)). Finally, when $\varepsilon = 1.0$ (Figure 25(d)), all the polygons are clustered together into one cluster. At this point, the value of $MinPolys$ has no effect on the clustering process.

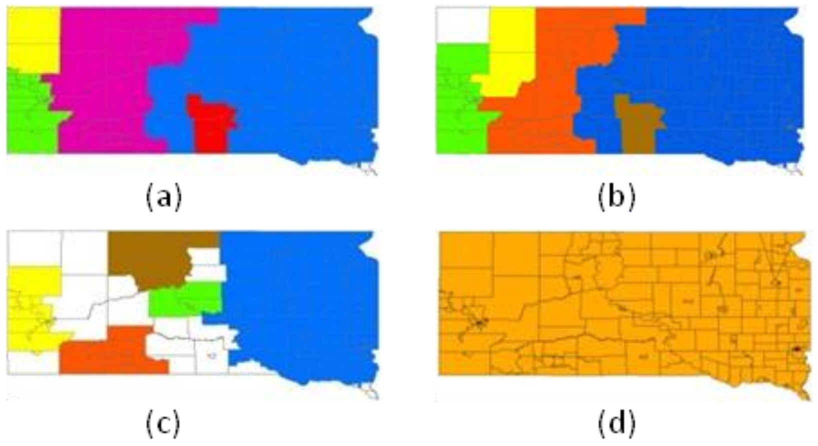


Figure 25: Results of clustering using P-DBSCAN (a) $\boldsymbol{\varepsilon = 0.65, MinPolys = 1}$ (b) $\boldsymbol{\varepsilon = 0.65, MinPolys = 2}$ (c) $\boldsymbol{\varepsilon = 0.65, MinPolys = 5}$ (d) $\boldsymbol{\varepsilon = 1.0, MinPolys = 5}$

The results obtained for DBSCAN and P-DBSCAN as shown above were compared using the compactness index. Once again we compare the results by computing the average compactness index of all the clusters formed at a given ε and $MinPolys = 5$. Figure 26 shows the results. The number above each bar represents number of clusters. P-DBSCAN produces clusters with a lower compactness ratio, except for in one case where DBSCAN produces greater number of small clusters. This implies that the clusters formed using P-DBSCAN are more compact than the clusters formed using DBSCAN.

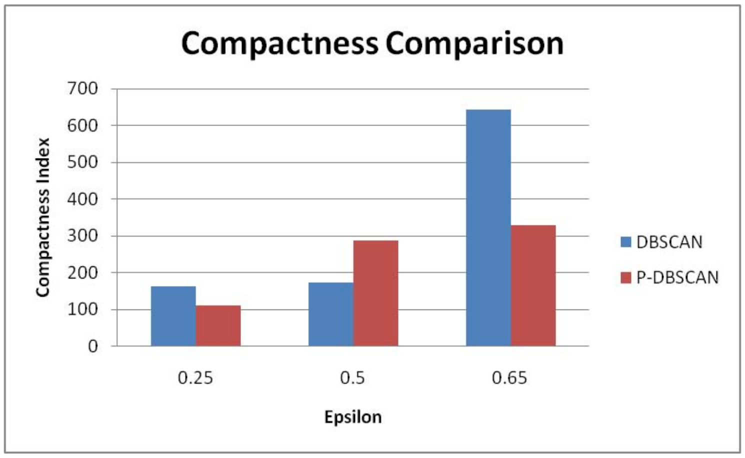


Figure 26: Compactness Ratio for clusters formed using DBSCAN and P-DBSCAN.

3.4.3 Summary of Experiments

Summarizing the results of our experiments, we make the following conclusions:

1) ε plays a major role in deciding the formation of the clusters. The smaller the ε the smaller will be the clusters. As we increase ε, there will always be a value at which all the polygons will be grouped into one cluster. Further, depending on the average size of the polygons and thus the average distance between polygon centroids, the value of ε should be adjusted accordingly. That is, if the polygons are large, then ε should be increased, and vice versa.

2) $MinPolys$ parameter plays an important role in deciding if a polygon is a core polygon or not. Compared to $MinPts$ in DBSCAN, additoinal information could be derived from the average neighborhood of a cluster to better select a value for $MinPolys$. For example, if the polygons are mostly rectangular such that each polygon is likely to have 3 or 4 neighbors, then setting $MinPolys$ = 5 might be too conservative, leading to many, small clusters. Further, by if the number of sectors of a polygon's neighborhood occupied by another polygon is generally large yet the number of neighboring polygons is low, then that indicates that polygons are surrounded by larger polygons. In that case, it might be more appropriate to set $MinPolys$ low.

3.5 Conclusion and Future Work

We have proposed a new clustering algorithm for clustering polygons. Our algorithm is based on the density-based clustering algorithm DBSCAN. While some concepts of DBSCAN are directly applicable for clustering polygons, concepts of core and border points as used in DBSCAN cannot be directly applied to define core and border polygons. Therefore, we re-define the concepts of core and border polygons. We introduce the concept of an outlier polygon, and a radial partition-based spatial neighborhood of a polygon which takes into account the topological properties of the polygons in addition to the density of the polygons in the dataset.

We also proposed using our modified Hausdorff distance function to compute the distance between the polygons while clustering them. Our distance function implicitly defines two polygons sharing a large extent of their boundaries to be close to each other. This is based on the intuitive concept of greater the sharing, more the similarity. However, we do not take into account that if the boundary is a country border, or a mountain range – a feature which may prohibit the clustering of the two polygons on either side together, then the distance should not be minimized. In our future research we will modify our distance function to take into account the type of the boundary between the two polygons.

Our comparison of the clustering results of DBSCAN and P-DBSCAN showed that more compact clusters are formed using P-DBSCAN. Thus our objective of producing compact clusters is satisfied by our proposed novel algorithm.

Currently, the clustering is done only on the basis of distance between the two polygons. In our future experiments, we plan to introduce the concept of spatial autocorrelation in the process of clustering to enhance the compactness of the clusters further. We will be performing multi-dimensional clustering, where more attributes of the polygons will be taken into account while clustering the polygons.

Acknowledgment

We would like to thank Tao Hong for his help in writing the programs to pre-process the datasets. We would also like to extend our thanks to Dr. David Marx for his guidance.

Publications

This chapter appears in the following:

1. Joshi, D., Samal, A., & Soh, L-. K. (2009). Density-Based Clustering of Polygons. *IEEE Symposium Series on Computational Intelligence and Data Mining*, (pp. 171-178). Nashville, TN.

Chapter 4: Density-Based Clustering of Polygons in the Presence of Obstacles

4.1 Introduction

Spatial clustering is the process of grouping similar objects based on their proximity to each other, or their relative density in space. It has numerous applications in spatial data mining, spatial data analysis, pattern recognition, image processing, market research etc. (Tung, Hou, & Han, 2001). The development of these algorithms has been an active area of research for the past several years (Ester, Frommelt, Kriegel, & Sander, 2000), (Han, Kamber, & Tung, Spatial clustering methods in data mining: A Survey, 2001). Most of these algorithms focus on clustering point data sets, and perform unsupervised classification of data objects. However, in geographic applications generally the space is divided into polygons such as census tracts, counties, states, watersheds, agro-economic zones, traffic analysis zones, etc. When point-based clustering techniques are applied to polygonal datasets, the current state-of-the-art in spatial clustering does not give accurate results (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b), (Joshi, Samal, & Soh, A Dissimilarity Function for Clustering Geospatial Polygons, 2009a), (Joshi, Soh, & Samal, Redistricting Using Heuristic-Based Polygonal Clustering, 2009c). To illustrate, an instance of an important application of polygonal clustering is the process of regionalization. Regionalization is the process of region building where smaller units (polygons) are grouped together into larger contiguous regions based on some attribute or criteria (Poone, 1997). Thus, regionalization produces clusters of polygons that are spatially compact and contiguous. If polygons are indeed represented as points and clustering is performed, the spatial information and relationships between polygons are not captured and utilized during the clustering process (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b), (Joshi, Samal, & Soh, A Dissimilarity

Function for Clustering Geospatial Polygons, 2009a), (Joshi, Soh, & Samal, Redistricting Using Heuristic-Based Polygonal Clustering, 2009c).

Furthermore, in the real world, obstacles such as rivers, lakes, mountains, and other man-made barriers such as political boundaries, are present that may affect how the grouping together of objects located close to each other should be clustered. For example, two adjacent counties that otherwise would belong the same cluster will no longer be clustered together if the shared boundary between the counties is also a part of the state boundary. Thus, the state boundary here acts as an obstacle that leads to the division of a big cluster into smaller clusters. In other words *obstacles may be defined as un-passable zones through which a path cannot be defined.*

Typically, clustering algorithms use the standard Euclidean distance in order to measure the spatial proximity of the objects. This assumes that there exists a straight line path between the two objects (Estivill-Castro & Lee, 2000a). However, this assumption fails in the presence of obstacles that prevent the traversing of the straight line paths between two objects. Thus, in this case, other distance functions that find the shortest feasible path between the objects in the presence of obstacles are required to measure the spatial proximity of objects.

Some of the spatial clustering algorithms have been extended to handle obstacles (Estivill-Castro & Lee, 2000b), (Wang, Rostoker, & Hamilton, 2004), (Zaïane & Lee, 2002), (Zhang, Wang, Wu, Fan, & Li, 2006) as spatial constraints. (See Section 4.2.1 for an overview of these algorithms.) However, all of these algorithms are designed for point datasets. By performing point-based clustering of polygons, the spatial and topological structure of the polygons would be lost. Furthermore, a major difference between point datasets and polygonal datasets is that while clustering the point datasets, the obstacles are always objects that are external to the point objects. Whereas, for the polygonal datasets, the obstacles may lie inside a polygon, and may even be shared by more than one polygon. For the algorithms described in Section 4.2.1 there is no easy method to handle such complications. The current state-of-the-art in the spatial clustering in the presence of obstacles thus cannot handle polygonal datasets with obstacles effectively.

This chapter presents two significant contributions to address the challenges described above. First, we define the *visibility relationship* between two polygons. The polygons may be completely visible to each other, partially visible, or invisible with respect to each other in the presence of obstacles. (The partially visible case is unique to polygons and is not applicable to point datasets.) These visibility relationships are defined with respect to the vertices of one polygon visible to the other polygon and vice-versa. Based on this number, we quantify the visibility between two polygons by computing the degree of visibility for the two polygons. These visibility relationships and the degree of visibility for the polygons are defined in Section 4.3.1. Second, we present a novel algorithm, P-DBSCAN+, for clustering polygons in the presence of obstacles. P-DBSCAN+ is based on the density-based clustering algorithm for polygons, namely, P-DBSCAN (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b), which in turn is based on the popular density-based clustering algorithm for point datasets, namely, DBSCAN (Ester, Kriegel, Sander, & Xu, 1996). Briefly, P-DBSCAN+ makes use of the polygonal density-based concepts defined in (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b) (c.f. Section 4.2.2) and use the obstructed Hausdorff distance function (c.f. Section 4.3.3) to find the obstructed ε-neighborhood of a polygon. The obstructed-facilitated Hausdorff distance is in turn computed using the visibility graph for the polygonal dataset. Furthermore, PDBSCAN+ considers three different types of obstacles: (1) point obstacles such as police stations and accident sites, (2) linear obstacles such as rivers, and (3) polygonal obstacles such as lakes and mountains.

In order to evaluate the effectiveness of our algorithm, we first demonstrate the behavior of P-DBSCAN+ on a synthetic dataset before evaluating its effectiveness in clustering census tracts with railroads, rivers, and lakes as obstacles in the city of Lincoln, NE. Furthermore, we also compare and contrast our algorithm with one of the existing point-based spatial clustering algorithm in the presence of obstacles, namely, DBCLuC. Our experiments show that P-DBSCAN+ outperforms the point-based clustering algorithms in handling polygonal datasets, and cases of partial visibility, by detecting clusters with overall visibility of 1.0. DBCLuC on the other hand clusters partially visible polygons with the completely visible polygons, and therefore produces clusters which may have some polygons that are not completely visible to other polygons within the cluster.

The rest of the chapter is organized as follows. Section 4.2 presents the related work giving a background on spatial clustering in the presence of obstacles, and the density-based concepts for polygons. Section 4.3 gives the basic definitions for our proposed approach, describes the obstructed distance function for polygons, and explains our algorithm in detail. Section 4.4 presents an application of our clustering algorithm. Finally, our conclusion and directions for future work are given in Section 4.5.

4.2 Related Work

In this section we present a brief introduction to the four main spatial clustering algorithms in the presence of obstacles – COD-CLARANS, AUTOCLUST+, DBCluC, and DBRS+. While COD-CLARANS follows a partitional clustering approach, AUTOCLUST+ is based on the principle of graph partitioning. Both DBCluC and DBRS+ are density-based clustering algorithms. Followed by which we present the density-based concepts for polygons as defined in (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b).

For our algorithm we have chosen to use a density-based approach for its advantages. First, it has the ability to discover clusters of arbitrary shapes such as linear, concave, and oval. Second, it does not require the number of clusters to be determined in advance. Finally, the density-based algorithms have been shown to be scalable to large databases. P-DBSCAN (the base algorithm (presented in (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b)) of the novel algorithm – P-DBSCAN+ presented in this chapter) extends DBSCAN (a well established density-based clustering algorithm) to cluster polygons instead of points by redefining the concepts of the neighborhood of a polygon, core polygon, border polygon, and noise polygon. The clustering is then performed based on the density-connectivity between two polygons leading to the polygons close to each other being clustered together, and thus resulting in spatially compact clusters.

4.2.1 Spatial Clustering in the Presence of Obstacles

COD_CLARANS (Tung, Hou, & Han, 2001) was the first obstacle constraint partitioning clustering method. It is a modified version of the CLARANS partitioning algorithm (Ng & Han, 2002) adapted for clustering in the presence of obstacles. The main idea is to replace the Euclidean distance function between two points with the obstructed distance, which is the length of the shortest Euclidean path between two points that does not intersect any obstacles. The calculation of obstructed distance is implemented with the help of several steps of preprocessing, including building a visibility graph, micro-clustering, and materializing spatial join indexes. After preprocessing, COD_CLARANS works efficiently on a large number of obstacles.

AUTOCLUST+ (Estivill-Castro & Lee, 2000b) is a version of AUTOCLUST (Estivill-Castro & Lee, 2000b) enhanced to handle obstacles. The advantage of the algorithm is that the user does not need to supply input parameter values such as the threshold distance or the number of clusters in order to detect the clusters. There are four steps in AUTOCLUST+. First, it constructs a Delaunay diagram (Delaunay, 1932). Then, a global variation indicator, the average of the standard deviations in the length of incident edges for all points, is calculated to obtain global information before considering any obstacles. Third, all edges that intersect with any obstacles are deleted. Finally, AUTOCLUST is applied to the planar graph resulting from the previous steps. When a Delaunay edge traverses an obstacle, the length of the distance between the two end-points of the edge is approximated by a detour path between the two points. However, the distance is not defined if no detour path exists between the obstructed points. AUTOCLUST+ algorithm inherits the limitation of AUTOCLUST algorithm, which builds a Delaunay structure to cluster data points with obstacles costly and is unfit for a large number of data.

DBCLuC (Zaïane & Lee, 2002), based on DBSCAN (Ester, Kriegel, Sander, & Xu, 1996), has been extended to handle obstacles. Instead of finding the shortest path between the two objects by traversing the edges of the obstacles, DBCLuC determines the visibility through obstruction lines. An obstruction line, as constructed during preprocessing, is an internal edge that maintains visible spaces for the obstacle polygons. Two points are considered to be visible to each other if the edge joining them does not intersect any obstruction line. After such preprocessing, DBCLuC is a very good density-based clustering approach for large datasets containing obstacles with many edges. However, constructing obstruction lines is very expensive for concave polygons, because the complexity is $O(v^2)$, where v is the number of convex vertices in obstacles.

DBRS+ (Wang, Rostoker, & Hamilton, 2004) extends the density-based clustering method DBRS (Wang & Hamilton, 2003) to handle obstacles. DBRS+ works by determining if both the points are within the same connected region. If yes, the distance between them as that computed by DBRS. However, if two data points belong to different connected regions, the distance between the two points is infinity. This distance function is known as the unobstructed distance function. Using the unobstructed distance function, DBRS+ then determines the unobstructed neighborhood of a point, following which it also uses the same approach as DBSCAN to detect the clusters. The worst-case time complexity of DBRS+ in the presence of obstacles is $O(nlog\ n+ nlogv+ nv'^2)$ where n is the size of the dataset, v is the number of vertices in the obstacles and v' is the total number of vertices in all local obstacles.

In this chapter, we compare P-DBSCAN+ to DBCluC as both are density-based clustering algorithms, though the latter is point-based. We do not include DBRS+ in this comparison, even though it is, as summarized above, also density-based. This is because DBRS+ also considers non-spatial attributes to help in the clustering process while P-DBSCAN+ does not, and suppressing the non-spatial attributes would essentially make DBRS+ the same as DBCLuC in terms of the results produced.

4.2.2 Density-Based Concepts for Polygons

Polygons are fundamentally different from points in that they are spread out in space, and factors that would have no effect on points—such as topology, shape and direction—have great influence on polygonal datasets. Also, if the polygons share boundaries, then two polygons sharing a larger extent of their boundary should be considered closer to each other as compared to two polygons sharing a very small portion of their boundaries. This conclusion follows from the observation that the closer the two polygons are to each other, more similar they will be in their characteristics (Poone, 1997). As a result of these factors, some of the density-based concepts for points do not directly apply to polygons. Mainly, the concept of a core polygon and its neighborhood are fundamentally different from that of a core point. Once a core polygon is defined, and the polygons that belong to its neighborhood, the same concepts of directly-density reachable, density-reachable, and density-connected for points can then be applied to polygons. The following density-based concepts for polygons have been defined in (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b) and are reproduced here for completeness and reference, especially needed when we discuss P-DBSCAN+ in Section 4.3.

[Definition 1] ε-neighborhood of a Polygon: The ε-neighborhood of a polygon *p*, denoted by $N_\varepsilon(p)$, is defined by $N_\varepsilon(p) = \{q \in D \mid dist(p,q)\} \le \varepsilon$, where *D* is the data set of polygons, and *dist(p,q)* is defined as the geographic distance between polygons *p* and *q* which may be computed using the distance functions such as the Hausdorff distance function or the Euclidean distance function.

[Definition 2] Radial Spatial Neighborhood of a Polygon: The neighborhood of a polygon can be further partitioned into sectors. That is, $N_\varepsilon(p) = \bigcup_{i=1}^{R} N_{\varepsilon,i}(p)$ such that R is the number of equal-size sectors radially partitioning the space around the polygon p. The definition of $N_{\varepsilon,i}(p)$ extends directly from the ε-neighborhood of the polygon, but only looks at the sector indexed by i. Figure 27 shows an example of the radial spatial neighborhood of the polygon p (shaded).

The radial spatial neighborhood of polygon p in Figure 27 is divided into 8 sectors: $S_1, \ldots, S_8$. As shown in Figure 27, $N_{\varepsilon,3}(p) = \{b, c\}$. Thus $\bigcup_{i=1}^{R} N_{\varepsilon,i}(p) = \{a, b, c, d, e, f, g\}$ which is the same as $N_\varepsilon(p)$.

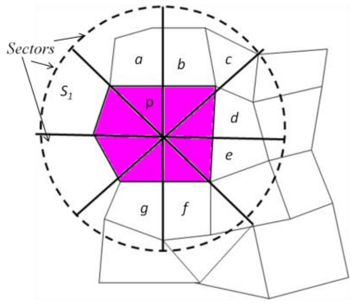


Figure 27: $\boldsymbol{R = 8}$ Radial spatial partitions of a polygon's neighborhood. Note that here the first sector is $\mathbf{S_1}$ as shown, and the ordering is clockwise. This is arbitrary for illustration purpose.

[Definition 3] Core Polygon: A core polygon c is defined as a polygon that has at least a minimum number of polygons (*MinPolys*) within its ε-neighborhood, and there are at least a minimum number of radial spatial partitions (*MinS*) that are non-empty, i.e. $\left[Count_{i=1}^{R}(N_{\varepsilon,i}(c) \neq \phi\right] \geq MinS$. For example, in Figure 28, if ε =1, *MinPolys* = 4 and *MinS* = 8, *p*, *o* and *q* are core polygons.

[Definition 4] Border Polygon: A border polygon b is defined as a polygon that has more than *R-MinS* of its radial spatial partitions empty, i.e. $\left[Count_{i=1}^{R}(N_{\varepsilon,i}(b) = \phi\right] > R - MinS$, where R is the total number of partitions. For example, in Figure 28 with ε =1, *MinPolys* = 4 and R = 8, and *MinS* = 8, *b* is a border polygon, since $\left[Count_{i=1}^{R}(N_{\varepsilon,i}(b)=\phi\right]=6$ and $R-MinS=0$.

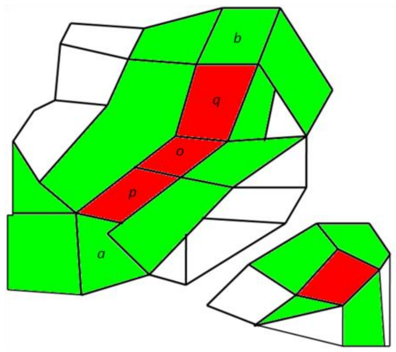


Figure 28: Synthetic set of polygons (Red – Core Polygon, Green - **ε**-neighborhood of the core polygons)

[Definition 5] Outlier Polygon: An outlier polygon is defined as a polygon that does not have any polygons within the threshold distance of ε.

[Definition 6] Directly Density-Reachable: A polygon *p* is *directly density-reachable* from a polygon *q* wrt ε, *MinPoly*, and *MinS* if

1) $p \in N_{\varepsilon}(q)$ and

2) *q* is a core polygon.

Directly density-reachable is symmetric for pairs of core polygons. In general, however, it is not symmetric if one core polygon and one border polygon are involved. For example, in Figure 28 polygon *a* is directly density-reachable from the polygon *p*, however polygon *p* is not directly density-reachable from the polygon *a*.

[Definition 7] Density-Reachable: A polygon *p* is *density-reachable* from a polygon *q* if there is a chain of polygons $p_1, \ldots p_n$ where $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i where {i = 1 to n-1}. In Figure 28 polygon *p* is density-reachable from polygon *q*.

[Definition 8] Density-Connected: A polygon *p* is density connected to a polygon *q* if there is a polygon *o* such that both, *p* and *q* are density-reachable from *o*. In Figure 28, polygon *a* and polygon *b* are density-reachable from polygon *o*, and thus are density-connected to each other.

[Definition 9] Cluster: A cluster C wrt. ε, *MinPoly*, and *MinS* is a non-empty subset of *D* satisfying the following conditions:

1) *Maximality*: $\forall p, q \mid p \in C, q \in D,$ q is density-reachable from *p*, then $q \in C$.
2) *Connectivity*: $\forall p, q \in C : p$ is density-connected to *q*.

4.3 Density-Based Clustering in the Presence of Obstacles

In this chapter we consider a set of polygons that have a set of obstacles present within the polygon cover. The goal is to detect density-based polygonal clusters in the presence of the obstacles. The main difference between the polygonal dataset and the point dataset is that the obstacles always lie exterior to the points while in the case of the polygons, the obstacles may lie within the polygons themselves, and may even be shared by two or more polygons. Thus while it is relatively simple to define the visibility relationship between two points—they are either visible to each, or they are not—such is not the case with polygons. Depending on the location of the obstacles, two polygons may only be partially visible to each other. In this chapter, we present a framework to model the polygons and the obstacles such that the visibility relationship between the polygons can be correctly detected, and the distance between the polygons can be accurately computed.

4.3.1 Preliminaries

Obstacles: Obstacles are defined as objects that serve as obstructions, and do not allow a path to be drawn across their body. In the literature obstacles are represented as polygons (Sack & Urrutia, 2000). However, in the real world there may be obstacles that do not have a polygonal shape, and instead may be a point or a linear feature instead. Examples of point obstacles include police stations, construction sites and accident sites. Examples of linear obstacles include rivers and railroad tracks. Examples of polygon obstacles include lakes, mountains and parks. We further take into account the area around the obstacle where the influence of the obstacle extends enough to make it impassable as well except along the border. We define this area as the zone of influence of the obstacle. For example, the immediate area around an accident site is also closed off to public access. This converts the point obstacle to a polygonal obstacle represented as a rectangle in our current framework. Furthermore, for linear obstacles such as rivers, a road cannot be constructed right at the banks of the river. A river flooding zone is designated and left as a gap between the road and the buffer. Taking this area into account, a river is converted into a long rectilinear polygon. In order to have a uniform representation of the different types of obstacles we model the zone of influence as a rectangular buffer area around an obstacle. Thus, in this framework each obstacle is represented as a polygon.

Visibility Graph of a Polygon: In computational geometry a visibility graph is defined as a graph whose nodes are the vertices of a polygon *P* and whose edges join pairs of vertices for which the corresponding line segment lies inside *P* (Sack & Urrutia, 2000). For convex polygons this will be a complete graph. However, more generally, there can be obstacles, sometimes called holes or islands (Kitzinger, 2003). The edges of the visibility graph in this case are represented between any two vertices if there are no obstacles present between them. For example, Figure 29 shows the visibility graph for the polygon *P* that has two obstacles O_1 and O_2 within its body.

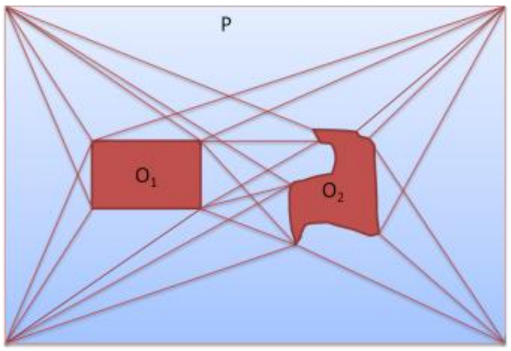


Figure 29: Sample visibility graph for a single polygon. O1 and O2 are obstacles while the lines constitute the visibility graph.

Polygonal s-t path: A polygonal s-t path between two vertices of the polygonal dataset is defined as a path from vertex *s* to vertex *t* consisting of a finite number of edges $(\{e_1,...,e_u\} \subset E)$ joining a sequence of vertices $(\{v_1,...,v_w\} \subset V)$.

Visibility Graph for a set of polygons in the presence of a set of obstacles: The visibility graph for a set of polygons $P = \{p_1,...,p_x\}$ in the presence of a set of obstacles $O = \{o_1,...,o_y\}$ is defined as a graph $VG = (V, E)$ where V represents the set of vertices of the polygons and the obstacles, and *E* is the set of edges which are defined by joining pairs of vertices such that

1) The edge $e_i \in E$ does not intersect any obstacle, i.e $e_i \cap O = \phi$.

2) The edges *E* lie within the polygon cover.

Figure 30 demonstrates an example of a visibility graph for a set of spatially contiguous polygons with a set of different types of obstacles. The red polygons, line, and point form different types of obstacles. The yellow buffer zones around each obstacle are the zones of influence of the obstacles respectively.

Note that this representation of an obstacle as a rectangle is only for convenience. A more appropriate representation would involve gradation in the degree of "obstruction" as one moves away from the center of the zone of influence. And as a result, that would also imply the visibility, as defined later, between two polygons through this zone can have different values depending on where the *s-t* path cuts across the zone. This will be a part of our future work.

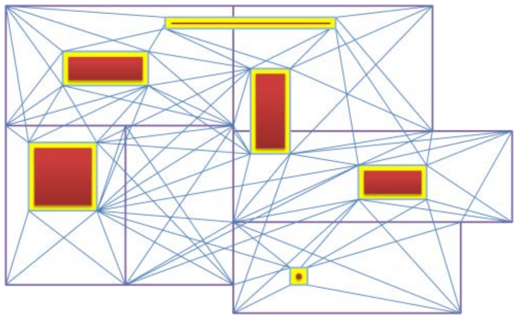


Figure 30: Sample visibility graph for a set of polygons in the presence of obstacles. The purple-outlined rectangles are polygons, the red polygons are obstacles with yellow-highlighted zones of influence, and the blue lines constitute the visibility graph.

Shortest s-t path length between two vertices: Each edge within the visibility graph $VG = (V, E)$ is assigned a weight equal to the length of the edge. In order to find the shortest *s-t* path length between any two vertices the Dijkstra's algorithm (Dijkstra, 1959) is applied on the visibility graph.

Reachability of a pair of vertices: A vertex v_i is said to be reachable from a vertex v_j, $(reachable(v_i, v_j))$, if there exists an *s-t* path between the vertices v_i and v_j in the visibility graph $VG = (V, E)$.

Visibility between polygons: Now, we further define the visibility relationship between polygon *A* and polygon *B* based on the reachability of the vertices of polygon *A* (V_A) with respect to the vertices of Polygon *B* (V_B), and vice versa. First, we define three basic types of visibility relationship between two polygons *A* and *B*:

1) **Complete Visibility –** If all the vertices of polygon *A* are reachable from all the vertices of the polygon *B*, and vice versa, then polygons *A* and *B* are said to be completely visible to each other (Figure 31). That is, $\forall a \in V_A, \forall b \in V_B \ \exists \ s-t \ path \ (a,b)$.

Figure 31: Polygons A & B are completely visible to each other

2) **Partial-Visibility –** If there exists at least one vertex of polygon *A* or polygon *B* that is not reachable from at least one vertex of polygon *B* or polygon *A* respectively, then the two polygons are said to be partially visible to each other.

   i. **Type A –** All the vertices of polygon *A* are reachable from a subset of the vertices of polygon *B* whereas not all vertices of polygon *B* are reachable from all the vertices of polygon *A* (Figure 32). That is, $\forall a \in V_A \ \exists b \in V_B \ s.t. \exists \ s-t \ path \ (a,b)$ and $\exists b \in V_B \ s.t. \neg\exists \ s-t \ path \ (a,b), \ a \in V_A$.

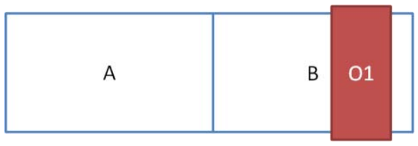


Figure 32: Polygon A and Polygon B are partially visible to each other under Type A partial visibility

ii. **Type B –** At least one vertex of polygon A is reachable from at least one vertex of polygon B, and vice versa (Figure 33). That is, $\exists a \in V_A, \exists b \in V_B \text{ s.t. } \exists s-t \text{ path}(a,b)$

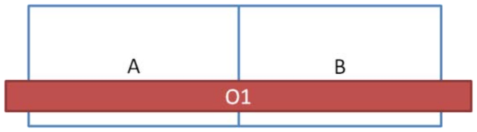


Figure 33: Polygon A and Polygon B are partially visible to each other under Type B partial visibility

3) **Invisible –** None of the vertices of Polygon *A* are reachable from any of the vertices of Polygon *B*, and vice versa (Figure 34). That is, $\forall a \in V_A, \forall b \in V_B \neg\exists\ s-t \text{ path } (a,b)$.

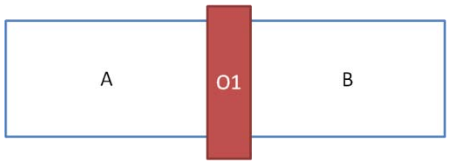


Figure 34: Polygon A and Polygon B are invisible to each other

***Degree of Visibility*:** The *number* or the *count* of vertices of polygon *A* visible to any vertex of polygon *B* (v_i) is represented as $count(v_i, A)$. Furthermore, due to the presence of obstacles the $count(v_i, A)$ may be different for each vertex of the polygon B. The maximum number of vertices of Polygon *A* visible to any vertex of polygon *B* is thus represented as $\max_{v_i \in V_B}(count\ (v_i, A))$.

$$Visibility\ (A \to B) = \frac{\max_{v_i \in V_B}(count\,(v_i, A))}{|V_A|}$$

$$Visibility\ (B \to A) = \frac{\max_{v_j \in V_A}(count\,(v_j, B))}{|V_B|}$$

$$Degree\,of\ Visbility\ (A,B) = Visibility\ (A,B) = \frac{Visibility\ (A \to B) + Visibility\ (B \to A)}{2}$$

While the relationship $Visibility\ (A, B)$ is a symmetric relationship between the two polygons *A* and *B*, *it is not a transitive relationship as it strictly depends on the visibility of the vertices of the two polygons*. For example, for the polygons *A* and *B* shown in Figure 32, the degree of visibility is computed as follows:

$$Visibility\ (A \to B) = \frac{4}{4} = 1$$

$$Visibility\ (B \to A) = \frac{4}{4} = 1$$

$$Visibility\ (A, B) = \frac{1+1}{2} = 1$$

For the polygons *A* and *B* shown in Figure 33, the degree of visibility is computed as follows:

$$Visibility\ (A \to B) = \frac{4}{4} = 1$$

$$Visibility\ (B \to A) = \frac{2}{4} = 0.5$$

$$Visibility\ (A, B) = \frac{1+0.5}{2} = 0.75$$

It should be noted that the degree of visibility of two polygons can also be based on area using the ratio of the visible area of a polygon to the total area of the polygon instead of using only the vertices of the polygons.

4.3.2 Distance Function for Polygons in the Presence of Obstacles

Once the visibility graph (*VG*) has been defined for the polygonal dataset in the presence of the given set of obstacles, the rest of the processing is performed using the shortest *s-t* path length between the vertices of the polygons computed using the visibility graph. Let polygon *A* be represented by a set of vertices $V_A = \{v_1,\ldots,v_n\}$, and polygon *B* be represented by a set of vertices $V_B = \{v_1,\ldots,v_m\}$.

The next step is to compute the pair-wise obstructed distance between the polygons within the dataset. In order to compute this distance we present a modified form of Hausdorff distance function that we term as the *obstructed Hausdorff distance (OHD).* For the two polygons A and B, the OHD ($d_{h\text{-}ob}$) is defined as follows:

$$d_{h-ob}(A,B) = \max(D_h(V_A,V_B), D_h(V_B,V_A)) \qquad (1)$$

where the Hausdorff distance function $(D_h(A,B))$ is computed as follows:

$$D_h(V_A,V_B) = \max_{a \in V_A}(\min_{b \in V_B} d_{VG}(a,b)) \quad (2)$$

where $d_{VG}(a,b)$ is the shortest *s-t* path length between the vertices *a* and *b* in *VG* computed using the Dijkstra's algorithm.

Intuitively, we expect the distance between two polygons with shared boundary to be less. However, the standard Hausdorff distance is defined on the set of points and does not incorporate any sharing of the boundary. In order to incorporate this, we define a new distance measure, called the *boundary adjusted obstructed Hausdorff distance* $(d_{h-ob'})$, that is inversely proportional to the length of the shared boundary between the two polygons, between two polygons *A* and *B* as follows:

$$d_{h-ob'}(A,B) = (1 - \frac{2S_{AB}}{S_A + S_B}) \times d_h(A,B) \quad (3)$$

where d_h is the obstructed Hausdorff distance function defined in Equation 1, S_A and S_B are the perimeter lengths of polygons *A* and *B* respectively, and S_{AB} is the length of their shared boundary. This distance, $d_{h-ob'}$, is smaller than the standard Hausdorff distance when two polygons have shared boundary, and becomes the standard Hausdorff distance when two polygons have no shared boundary, i.e., when $S_{AB} = 0$. We use twice the shared distance in the definition to balance the effect of the denominator. This transformation of the Hausdorff Distance function to incorporate the shared boundary length between two polygons has been previously successfully used in clustering polygons in (Joshi, Samal, & Soh, A Dissimilarity Function for Clustering Geospatial Polygons, 2009a) and (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b).

4.3.3 Density-Based Concepts for Polygons in the Presence of Obstacles

While most of the density-based concepts for polygons presented in Section 4.2 still hold true in the presence of obstacles, some changes need to be made in the framework to take into account the visibility relationship between polygons. We have redefined the ε- neighborhood of a polygon to accommodate the visibility properties. The new definition is presented below.

[Definition 1'] *Obstructed ε- neighborhood of a polygon*: The obstructed ε- neighborhood of a polygon *p* denoted by $N_{\varepsilon-ob}(p)$, is defined by

$N_{\varepsilon-ob}(p) = \{q \in D \mid d_{h-ob}(p,q) \leq \varepsilon \text{ } and \text{ } Visibility\ (p,q) \geq \alpha,\ 0 \leq \alpha \leq 1$, where D is the data set of polygons, ε and α are user-defined parameters, and $d_{h-ob}(p,q)$ is the obstructed Hausdorff distance function between polygons *p* and *q*, and is defined in Section 4.3.3.

[Definition 2' to 8'] Given the new obstructed ε- neighborhood of a polygon, the definitions **[Definition 2' to 8']** for density-based concepts for polygons in the presence of obstacles remain the same as **[Definitions 2 to 8]** with the replacement of $N_{\varepsilon}(p)$ with $N_{\varepsilon-ob}(p)$.

However, since the density-connectivity relationships are all transitive in nature, and the visibility relationship between the polygons is not transitive between polygons, we need a new concept to capture visibility-connectedness for polygons defined as follows:

[Definition 9'] *Visibility-connectedness:* Two polygons *p* and *q* are said to be visibility-connected if the $Degree\,of\ Visbility\ (p,q) \geq \alpha$, where α is a user defined parameter.

[Definition 10'] *Density-connected Cluster of polygons in the presence of obstacles*: A *cluster C* wrt. ε and α is a non-empty subset of *D* satisfying the following conditions:

1) *Maximality*: $\forall p,q \mid p \in C, q \in D$ if *q* is density-reachable from *p*, then $q \in C$.

2) *Connectivity*: $\forall p,q \in C$, *p* is density-connected to *q*.

3) *Visibility:* $\forall p,q \in C$, *p* is visibility-connected to *q*.

4.3.4 P-DBSCAN+ Algorithm

In order to detect density-connected clusters of polygons in the presence of obstacles, we first need to pre-process the dataset using the Pre-Processing module shown in Figure 35, and then apply the P-DBSCAN+ algorithm shown in Figure 36. The Pre-Processing module first forms the visibility graph for the polygons being clustered in the presence of the given set of obstacles, and then computes the pair-wise obstructed distance between all the polygons in the given dataset. This pair-wise distance is stored in a text file which becomes an input to the P-DBSCAN+ algorithm.

P-DBSCAN+ follows an underlying process similar to DBSCAN. Cluster ID (CID) is generated, and assigned the value 1. A polygon *p* is selected randomly from the set of polygons *P* and checked to see if it has been assigned to a cluster already. If *p* has not been assigned to a cluster, then the *ExpandCluster* routine is called. As in DBSCAN, *ExpandCluster* is where the cluster assignments are done.

The *ExpandCluster* routine checks to see whether *p* is a core polygon (cf., Definition 3’). For this, it first computes *p*’s obstructed ε- neighborhood $N_{\varepsilon-ob}(p)$ (cf., Definition 1’). Next, if the $N_{\varepsilon-ob}(p)$ contains at least *MinPoly* polygons and they cover at least *MinS* radial spatial partitions of the polygon (cf., Definition 2’), then *p* is classified as a core polygon. Next, *p* is assigned to the cluster CID if $\forall q \in C_{CID}, \textit{Degree of Visibility}\ (p,q) \geq \alpha$. Followed by which, the *ExpandCluster* routine is called recursively for each of the neighbors of polygon *p*. Figure 36 presents our proposed P-DBSCAN+ Algorithm.

```
Pre-Processing
Input: Set of Polygons P, Set of Obstacles O
Output: Pair-wise boundary adjusted Obstructed Hausdorff Distances for P
Construct the visibility graph VG for P taking O into account as follows:
    $\forall v_i, v_j$ s.t. $v_i \in V_P$ or $v_i \in V_O$ and $v_j \in V_p$ or $v_j \in V_O$ and $i \neq j$ add an edge from
    $v_i$ to $v_j$ to VG if:
        1) $(v_i, v_j) \cap O = \phi$.
        2) $(v_i, v_j)$ lies within the polygon cover P.
For each pair of vertices in VG, compute the shortest s-t path length using the Dijkstra's algorithm.
For each pair of polygons in P compute the pair-wise boundary adjusted Obstructed Hausdorff Distance using the formula presented in Section 3.2.
```

Figure 35: Pre-Processing algorithm.

```
P-DBSCAN+
Input: P, $\varepsilon$, $\alpha$, MinPoly, MinS, Pair-wise boundary adjusted Obstructed Hausdorff Distances for P
Output: Set of Clusters
Set CID = 1
For each polygon p in P
    If ClusterID(p) is UNCLASSIFIED then
        Call ExpandCluster
        If ExpandCluster then
            Increment CID
        End If
    End If
End For
```

```
ExpandCluster
Input: p, CID, $\varepsilon$, $\alpha$, MinPoly, MinS
Output: True or False
    If p is surrounded by polygons in at least MinS radial spatial partitions then
    $N_{\varepsilon-ob}(p)$ = Get the obstructed ε-Neighborhood of p.
    For each pn in $N_{\varepsilon-ob}(p)$
        If $Degree\ of\ Visibility\ (pn, p) > \alpha$
            Remove pn from $N_{\varepsilon-ob}(p)$
        End if
    End for
    If $N_{\varepsilon-ob}(p)$ contains at least MinPoly polygons then
        If $\forall q \in C_{CID}$, $Degree\ of\ Visibility\ (p, q) \geq \alpha$
            Set the ClusterID of p to CID
                For each pn in $N_{\varepsilon-ob}(p)$
                Call ExpandCluster (pn, CID, $\varepsilon$, $\alpha$, MinPoly, MinS)
                End For
                Return True.
        Else return False.
            End IF
    Else return False.
    End IF
Else return False.
End If
```

Figure 36: P-DBSCAN+ clustering algorithm.

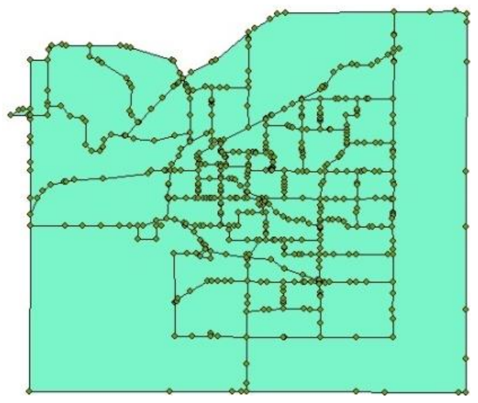
4.3.5 Computational Complexity of P-DBSCAN+

The computational complexity of pre-PDBSCAN+ is $O(n^2)$ as all the three steps of the algorithm – construction of visibility graph, finding shortest path lengths using Dijkstra's algorithm, and computation of pair-wise boundary adjusted obstructed Hausdorff distance for the polygons in the dataset – require $O(n^2)$ number of computations, where n is the number of vertices of the polygons. In order to reduce the computational complexity it is important to reduce the vertices of the polygons. In many practical applications, the number of vertices of polygons can be greatly reduced without compromise in the quality of data using the Douglas-Peucker polygon simplification algorithm (Peucker & Douglas, 1975) is used. For example, Figure 37(a) shows the census tracts for the city of Lincoln, Nebraska. Originally, the dataset has 55 polygons and the total number of vertices of all the polygons is 1211. After the application of the Douglas-Peucker algorithm the polygons are simplified and have only 408 vertices. The simplified polygons are presented in Figure 37(b). Furthermore, an optimized version of the Dijkstra's algorithm can be used.

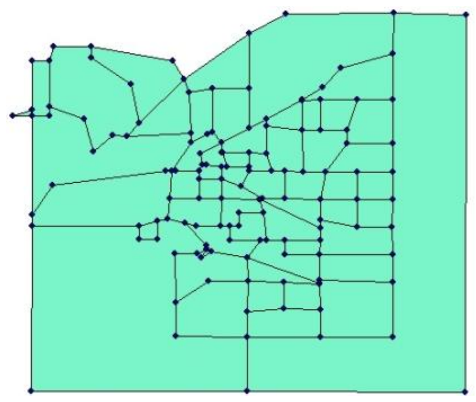
With the pair-wise boundary adjusted obstructed Hausdorff distance for the polygons pre-computed, and with the use of an indexing structure such as a R*-tree, the computational complexity of P-DBSCAN+ will be the same as that of DBSCAN, i.e. $O(n \log n)$.

4.3.6 P-DBSCAN++

In this section we propose an alternate version of the P-DBSCAN+ algorithm that allows the user to detect strong clusters first, i.e. clusters with degree of visibility 1.0, followed by the detection of weak clusters, i.e. clusters with degree of visibility between 0 and 1. The core algorithm remains the same as P-DBSCAN+ (Figure 36). The idea is to first apply P-DBSCAN+ with $\alpha = 1.0$. The user is not given an option here to select an alternate α. Once the results are obtained for the first application of P-DBSCAN+, the user is now allowed to select any α, and re-apply P-DBSCAN+ on the polygons that were not assigned to any cluster previously. Thus the second iteration of P-DBSCAN+ with a weaker α will allow the user to detect weaker clusters. Using this approach we give the user the freedom to detect the maximum number of clusters while still retaining some clusters with visibility 1.0.



(a)



(b)

Figure 37: (a) Lincoln, NE census tracts – 55 polygons with 1211 vertices. (b) Simplified Lincoln, NE census tracts using the Douglas-Peucker algorithm – 55 polygons with 408 vertices.

4.4 Experimental Analysis

In this section we describe the performance of our algorithm P-DBSCAN+ on a synthetic dataset for 110 polygons and on a real dataset comprising of the census tracts of Lincoln, NE. Obstacles have been added to both the datasets. We also compare the results obtained by P-DBSCAN+ with the results of P-DBSCAN and DBCLuC. P-DBSCAN is the parent algorithm of P-DBSCAN+ in the sense that it does not handle obstacles as constraints. The reason for this comparison is to show the importance of taking the obstacles into consideration while forming clusters of spatial polygons. DBCLuC, on the other hand, is a density-based clustering algorithm for point datasets in the presence of obstacles. As the case of partial visibility does not exist in point datasets, DBCLuC cannot handle with such cases.

4.4.1 Experiment with Synthetic Dataset

For the first set of experiments, we use a synthetic dataset of 110 polygons and 5 obstacles (Figure 38). As described before, all the obstacles (polygons, lines and points) are represented by taking their zone of influence into account and hence are polygons in our approach.

First, P-DBSCAN clusters polygons based on the input parameters of ε, *MinPoly*, and *MinS* without taking into account the obstacles that may be present. Thus with a large enough ε (e.g., = 200), we find that all the polygons get clustered together as shown in Figure 39.

DBCLuC, on the other hand, takes a point representation of the polygons. We use the most rudimentary representation of a polygon as a point, i.e. the centroid of each polygon. Furthermore, the point-based clustering approaches only detect whether two points are visible to each other or not before clustering them together. Thus, applying DBCLuC to our synthetic dataset, once again with a large enough ε (e.g., = 200) we find that the polygons get split into two clusters (Figure 40) where none of the clusters are strong clusters, i.e. the overall degree of visibility for both the clusters is less than 1.0.

Next, we apply P-DBSCAN+ to the synthetic dataset. P-DBSCAN+ clusters polygons based on the input parameters of ε, α, *MinPoly,* and *MinS*. The input parameter α plays a key role here and allows the user to control the purity of visibility within the clusters. Thus, if $\alpha = 1.0$ and $\varepsilon = 200$ the result obtained by P-DBSCAN+ will be two clusters with a set of polygons all sharing the linear feature detected as *outliers* (Figure 41). The two clusters detected have the overall degree of visibility of 1.0. *It should be noted that DBCLuC or any other point-based clustering algorithm will never be able to detect the outliers as found by P-DBSCAN+.*

Finally, if we are to apply P-DBSCAN++ to the synthetic dataset where the first iteration is done with $\alpha = 1.0$ and the second iteration is done with $\alpha = 0.5$, we find that three clusters are detected (Figure 42). While the two clusters detected in the first iteration are the same as the ones detected previously with overall degree of visibility of 1.0, the third cluster detected is composed of the polygons that share the linear obstacle and were previously classified as outliers.

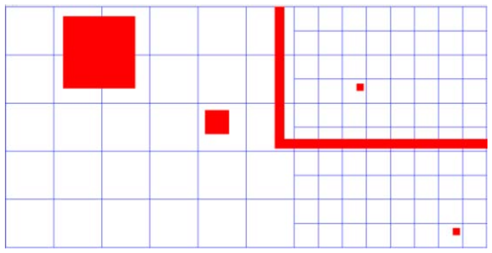


Figure 38: Synthetic dataset with 110 polygons and 5 obstacles

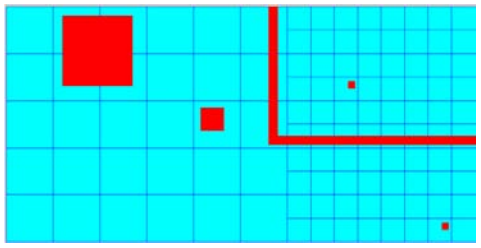


Figure 39: Result of clustering using P-DBSCAN, i.e. without taking the obstacles into consideration ε = 200

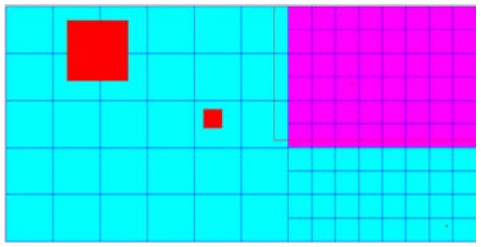


Figure 40: Result of clustering using DBCLuC with ε = 200.

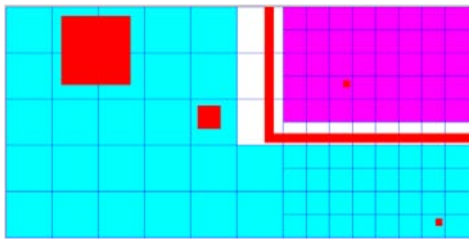


Figure 41: Result of clustering using P-DBSCAN+ with ε = 200 and. α = 1.0

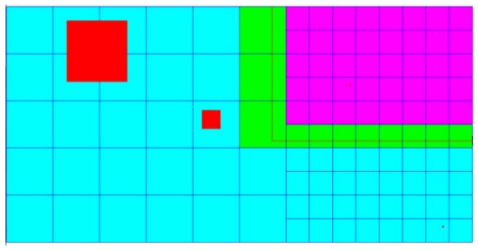


Figure 42: Result of clustering using P-DBSCAN++ with ε = 200 and α = 1.0, 0.5

4.4.2 Experiment with Lincoln Census Tract Dataset

For the second set of experiments, we use the census tract dataset for the city of Lincoln, NE. The dataset consists of 54 polygons and 3 obstacles. The obstacles are in the form of a rail road track, a stream, and a park (Figure 43). The zone of influence is the buffer area around each obstacle where its influence is extended. Taking the zone of influence of each obstacle following the approach described earlier in Section 4.3.1, the new representation of the obstacles is shown in Figure 44.

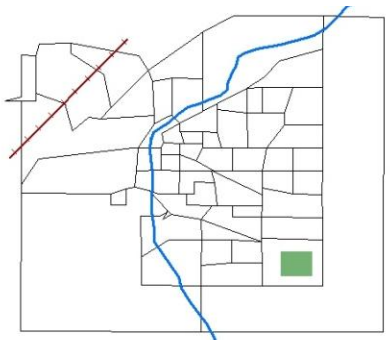


Figure 43: Census tract dataset of the city of Lincoln, NE with 55 polygons and 3 obstacles.

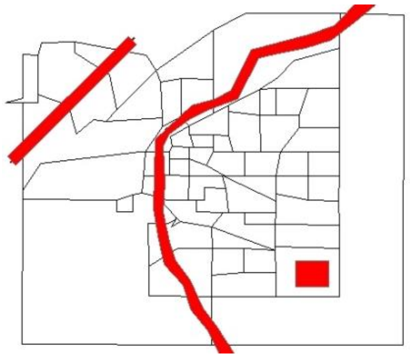


Figure 44: Census tract dataset of the city of Lincoln, NE with obstacles modeled as rectangular polygons

The clustering together of census tracts in the presence of obstacles makes sense because for certain applications. For example, when determining location of ambulance services to guarantee service time, one must account for the railroad tracks since the crossings may be blocked. We applied both DBCLuC and P-DBSCAN+ to this dataset. The result obtained by the DBCLuC algorithm is shown in Figure 45. As mentioned before, DBCLuC is a point-based clustering algorithm; thus, we represent the polygons by their centroids, i.e., points, for DBCLuC. Once again, as the case of partial visibility does not exist among points, the clusters were formed without taking into consideration that while the centroid may be visible to the centroid of another polygon, when in fact the entire polygon is not visible. Thus, the final clusters produced have polygons as their members that were being cut through by the obstacles leaving some portions of these polygons completely invisible to the rest of the cluster. Therefore, not every polygon is completely visible to every other polygon within the same cluster.

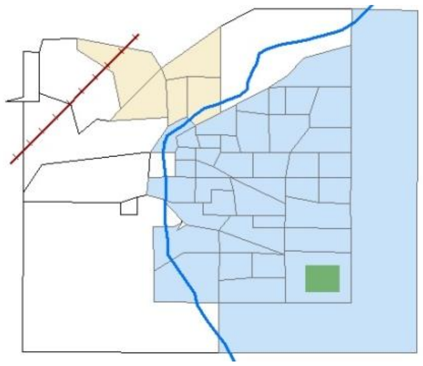


Figure 45: Result of clustering using DBCLuC.

The result obtained by P-DBSCAN+ is shown in Figure 46. P-DBSCAN+ is well equipped to handle cases of partial visibility and therefore successfully detects clusters with where every polygon is completely visible to every other polygon within the same cluster. Furthermore, it can be seen that in comparison to the results obtained based on the point representation of the polygons, our approach does a better job at detecting the density-based clusters. In Figure 44, the large inverted L-shaped polygon at the right end of the dataset is added to the same cluster as the other smaller polygons towards its left. This is because the centroid distance between the polygons was small. However, by computing the distance using our proposed distance function, this polygon is no longer clustered together with the smaller polygons as can be seen in Figure 45. It is important to note that for real geographic datasets, in order to detect clusters that may be formed due to some linear feature, the input parameter of *MinS* provided to P-DBSCAN+ should be set to 1 or a maximum of 2. This is because for a cluster that is linear in shape, the core polygons will not be surrounded by polygons in every radial spatial partition.

Finally, when we apply P-DBSCAN++ to this dataset, the result obtained is shown in Figure 47. It can be observed that two more clusters are added to the total number of clusters originally detected by P-DBSCAN+. These two clusters are unique because they are composed of polygons that share the same obstacles respectively. Having such clusters can provide further insight to the decision makers in practical applications to assign these polygons to the appropriate clusters. For example, one could argue that all the census tracts surrounding the railroad should be clustered as one because they share the same characteristics with respect to the railroad, in a way analogous to how watershed areas are clustered along a river. Another alternative would be to split each of these polygons into two or more smaller polygons so that each portion can be designated to the appropriate cluster so that every polygon is completely visible to every other polygon within the same cluster. Finally, the large polygon (with no color, to the right of the map) is left out as an outlier that may be assigned to a cluster of its own.

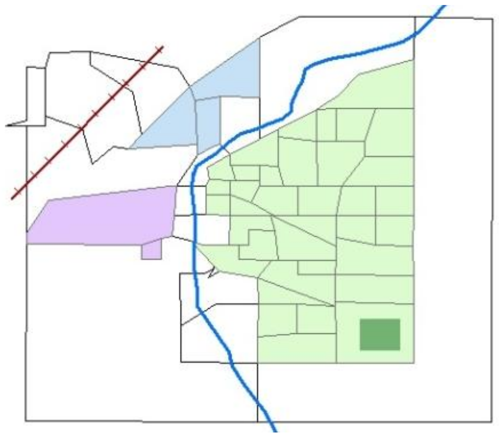


Figure 46: Result of clustering using P-DBSCAN+ with α = 1.0

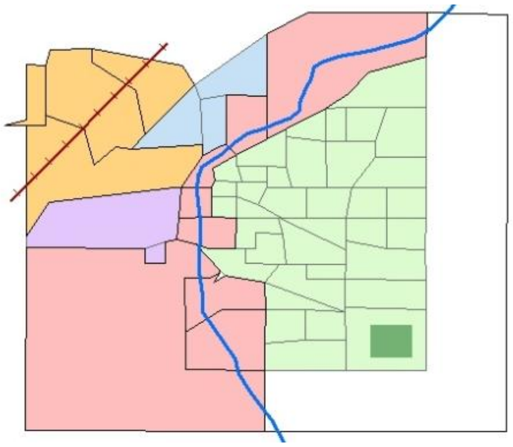


Figure 47: Result of clustering using P-DBSCAN++ with α = 1.0, 0.5

4.5 Conclusion and Future Work

In this chapter, we have proposed our research and investigation into spatial clustering of polygons in the presence of obstacles. First, we have defined there types of visibility relationship between two polygons: complete visibility, partially visibility, or invisibility. We have also defined the degree of visibility between two polygons that quantifies the visibility relationship in the presence of obstacles. Using the visibility relationship, we have extended the P-DBSCAN algorithm to the P-DBSCAN+ algorithm that clusters polygons in the presence of obstacles. We have also proposed a variant P-DBSCAN++ that allows clusters of complete visibility and partial visibility to be detected. Our experiments compared our algorithms with the point-based, density-based DBCLuC algorithm in a synthetic dataset and a real-world census tract dataset. We have demonstrated that, from the results, both P-DBSCAN+ and its variant are able to handle obstacles well while the variant P-DBSCAN++ is able to provide a more "complete" clustering by also detecting weaker clusters. Both algorithms are also better than DBCLuC, while more studies will need to be conducted to ascertain the validity of our results with full confidence.

So far we have applied our algorithm to a set of spatially contiguous polygons. The algorithm and the pre-processing of the data may easily be extended to a set of non-contiguous polygons by drawing the visibility graph for each individual polygon, and in order to find the distance in between the polygons, we simply find the Euclidean distance between the vertices.

We have treated obstacles as impassable zones in this chapter unless there is a facilitator present that allows one to define a path through the obstacle. However, in many cases one may cross through an obstacle such as a mountain or a lake with additional cost. As part of our future work, we will define a new distance function that takes into consideration this cost function for the various obstacles. Furthermore, based on the different types of obstacles the weight of the cost function will be varied as it may be easier to go through one type of obstacle as compared to another. We will also take into account the gradation in the degree of "obstruction" as one moves away from the center of the zone of influence. And as a result, that would also imply the visibility, between two polygons through this zone, can have different values depending on where the *s*-*t* path cuts across the zone.

In addition to obstacles, another category of objects may be present within the dataset that may influence the result of the proximity-based or density-based clustering algorithms. These are known as *facilitators*. Facilitators are objects that help in reducing the distance between two spatial objects, and therefore making them closer to each other. Examples of facilitators include highways, bridges, etc. Thus while the obstacles may split a cluster into two or more smaller clusters, the presence of facilitators may lead to the unification of two or more smaller clusters into a single unified cluster. For example, if there is no path between two polygons because of the presence of a river between them, both the polygons will belong to different clusters. However, if a bridge is present on the river connecting the two polygons together, they may now belong to the same cluster. We also plan to include the facilitators within our framework.

Publications

This chapter appears in the following:

1. Joshi, D., Samal, A., & Soh, L-. K. (under preparation). Polygonal Spatial clustering in the Presence of Obstacles and Facilitators, to be submitted to *Transactions in GIS*.

Chapter 5: Constraint-Based Clustering of Polygons

5.1 Introduction

Redistricting is the process of dividing a geographic space or region of spatial units often represented as polygons into smaller subregions or districts. In other words, it can be viewed as a set partitioning problem, i.e. the problem is to cluster the entire set of spatial polygons into groups such that a value function is maximized (Altman, 2001). Because of the spatial properties involved, redistricting is akin to spatial clustering. At the same time, as the most common use of these districts is to facilitate some form of jurisdiction, redistricting often involves satisfying or conforming to constraints that represent policies, laws and regulations. Typical spatially-flavored constraints are spatial contiguity and compactness, while an example of domain-specific constraint is uniform population (or resource) distribution.

Spatial clustering deals with spatial data that is generally organized in the form of a set of *points* or *polygons*. Most spatial clustering algorithms proposed in the literature focus on clustering point data (Han, Kamber, & Tung, Spatial clustering methods in data mining: A Survey, 2001). However, when applying these algorithms to cluster polygons instead of points, these algorithms fall short of giving accurate results (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b). The main cause of the inadequacy is that in comparison to polygons, points are relatively simpler geographic objects. Polygons, especially in the geographic space, share spatial and topological relationships and cannot be accurately represented as points. For example, two polygons may share boundaries with each other, or may cover different amounts of area. None of these conditions can be captured in point datasets. Redistricting is thus a polygonal spatial clustering problem as most of the space around us is divided into polygons, e.g. states, counties, census tracts, blocks, etc.

Furthermore, while clustering is a form of unsupervised learning, redistricting requires the use of some form of domain knowledge. Efficient use of this available information during the process of clustering can significantly enhance the quality of the clusters. Use of constraints in clustering is widely examined in data mining. Examples of constraint-based clustering algorithms are COP-KMEANS (Wagstaff, Cardie, Rogers, & Schroedl, 2001), C-DBSCAN (Ruiz, Spiliopoulou, & Ruiz, 2007), etc. However, these algorithms are all point-based. Constraints applied during the process of clustering can be of two types – instance-level constraints and cluster-level constraints. Instance-level constraints are applied to individual objects being clustered. Examples of instance-level constraints are must-link and cannot-link constraints (Davidson & Ravi, 2005). Cluster-level constraints on the other hand, are applied to the cluster as a whole. Examples of cluster-level constraints are averaging or summation constraints (Davidson & Ravi, 2004). For example, the sum of the population of a cluster must be less than or equal to x. It has been proven that satisfying such cluster-level constraints in the clustering process is NP-hard (Altman, 2001).

In this chapter we present a suite of clustering algorithms for clustering spatial polygons in the presence of constraints. The core algorithm, called the Constrained Polygonal Spatial Clustering (CPSC) algorithm, is designed to solve the problem when the constraints are hard and inviolable. We further propose two extensions of CPSC, namely, CPSC* and CPSC*-PS (i.e. CPSC* with Polygon Split). CPSC* is designed to handle soft constraints, while CPSC*-PS is a further extension to allow a polygon to be split during the clustering process using an underlying tessellation in order to improve the quality of the clustering results. The uniqueness of these algorithms is that they make use of the spatial and topological relationships between the polygons as well as the domain knowledge present in the form of constraints to cluster polygons using an A* search-like underlying process. Briefly, the core algorithm CPSC is divided into three main steps: 1) select seeds, 2) decide the best cluster to grow, and 3) select the best polygon to be added to the best cluster. Several novel strategies of the algorithm include:

- *Use of heuristic functions to apply the constraints during clustering*. A heuristic function (F) has two components: (1) the distance function (H) that measures the distance of the current state of the cluster from the desired goal, and (2) the cost function (G) that measures the reduction in the flexibility of the growth of all the other clusters. The use of these heuristic functions facilitates efficient use of agglomerative type cluster-level constraints.
- *Integration of constraints in seed selection*. Instance-level and cluster-level constraints from the domain are used from the outset in seed selection. By applying the constraints and selecting the seeds using the heuristic functions we make the algorithm more robust to order dependency and poor initial seeding.
- *Selection of the best cluster to grow*. At the beginning of each iteration CPSC selects the best cluster to grow based on the heuristic function (F) that approximates the level of "need-to-grow" for each cluster; that is, the cluster with the greatest need is selected to be grown next.
- *Selection of the best polygon to be added to the best polygon.* Once the best cluster has been selected to grow, the best polygon in terms of the level of "reduction of flexibility" is selected using the heuristic function F'; that is, the polygon with minimal impact on the growth of surrounding clusters is chosen to be added to the best cluster.
- *The polygons are allowed to move from one cluster to another.* As the growth process of a cluster follows a greedy approach, every cluster selects the polygon that minimizes its need at that stage. A cluster may decide that the polygon which has already been assigned to another polygon is the best polygon that meets its need. The move of the polygon from its original cluster to the new cluster is allowed by CPSC in the special case when a new cluster has no unassigned polygon present in its neighborhood.

Based on the same underlying process as CPSC, CPSC* finds a solution by allowing the user to prioritize the constraints. Further, it relaxes the constraints to allow un-clustered polygons to be assigned to clusters even though they would have violated the original constraints. CPSC* also has a deadlock detection and breaking mechanism that ensures convergence of the algorithm. CPSC*-PS (i.e., with polygon split) further improves the quality of the clusters produced by CPSC*. It uses the underlying structure within each polygon to split it into smaller polygons, which can then be assigned to different clusters, thus taking the clusters produced by CPSC* a step closer to the desired target state.

For our comparative and validation study, we apply the CPSC suite to two widely used redistricting problems: *congressional redistricting* and *formation of school districts*. We compare the results of CPSC for the congressional redistricting problem with three other techniques based on graph partitioning (Bodin, 1973), simulated annealing (Macmillan, 2001), and genetic-based algorithms (Bacao, Lobo, & Painho, 2005). Congressional redistricting has been inflicted traditionally with issues such as Gerrymandering (Hayes, 1996) and unequal population distribution. In our study, we find that our algorithm outperforms the other three algorithms by producing districts that have almost equal population and are spatially compact. We then applied CPSC* to the problem of school districting which is a task that is frequently performed to assess the distribution of resources and delegation of authority. Finally, in order to validate the CPSC*-PS algorithm, we have applied it to a sample dataset.

The chapter is organized as follows. 5. 2 discusses other redistricting algorithms. Section 5.3 presents the CPSC algorithm, and its two extensions. Section 5.4 describes the application domains and implementation of the CPSC algorithm suite in each domain. Section 5.5 covers the experimental analysis of our algorithms. Finally Section 5.6 gives our conclusions and the directions for future work.

5.2 Related Work

Redistricting is essentially an optimization problem where the global optimum solution is difficult to find. This is because the size of the solution space can be enormous. A simple brute force search through all the possible solutions is impractical especially when the dataset size increases.

As a result, the problem of redistricting has been considered to be difficult to solve precisely and efficiently. Moreover, due to the size of the real datasets, most of the current techniques used for automated redistricting resort to unproven guesswork (Altman, 2001) and random selection, and are therefore inefficient and may not be accurate.

Several meta-heuristic approaches have been proposed in the past to solve this problem. These meta-heuristic approaches are often based on genetic algorithms, simulated annealing, or graph partitioning techniques. While all of these algorithms work with polygonal datasets, they do not exploit either the spatial properties of the polygons themselves or the nature of the geographic space. In this section we give an overview of different approaches that have been implemented to solve redistricting problems. The different approaches discussed here are graph partitioning, simulated annealing and genetic algorithm based redistricting methods. The implementation of these methods shown here is for the congressional redistricting problem. We have highlighted their advantages and disadvantages. The results of these algorithms are compared with CPSC in Section 5.5.1.

5.2.1 Graph Partitioning

The problem of partitioning a geographic area into a collection of contiguous, approximately equal population districts can be viewed as a graph partitioning problem. The graph is formed by representing each polygon within the dataset as a node, and the polygons that share boundaries are connected by an edge. Furthermore, each node is assigned a weight which is equal to the population of the polygon. The problem is now to divide the graph into a fixed number of sub-graphs or clusters such that the sum of the weights of the nodes within each cluster is equal, and each cluster is connected. The outline of the *graph partitioning algorithm for congressional redistricting* proposed by (Bodin, 1973) is as follows.

A label is assigned to each node in the graph. This label has three components: the cluster number to which the node belongs (r), the weight of the cluster(s), and the predecessor of the node in the graph (t). The weight of the cluster is the sum of the populations of the nodes assigned to that cluster. Initially every node is assigned the label – $[0, \infty, 0]$, indicating that the $r = 0, s = infinity$, and the $t = 0$. The seeds of the clusters are selected randomly. Each seed is assigned to a separate cluster, and its label is changed to $[j, x, -1]$ where j is the cluster number ($r = j$) and x is the population of the seed ($s = x$). And, as each seed forms the root of a cluster, it does not have any predecessor. Therefore $t = -1$. Subsequently a pass is made through all the nodes in the graph and each node is assigned to a cluster based on the weight of the cluster, where the weight of the cluster is equal to the total population of the cluster, and the predecessor of the node.

Step 2 of the algorithm takes the clusters produced in Step 1 and improves them by exchanging nodes between clusters. Spatial contiguity is preserved during the exchange process.

The advantages of this approach are that it is computationally fast, and it presents the user with several potentially useful plans. However, there are several disadvantages with this approach. 1) While this procedure is extremely fast computationally, it does not always terminate at an optimal solution. 2) The random selection of seeds may lead to the development of poor plans. 3) There are no guidelines provided in this methodology to select the best plan. 4) This method does not work well when the number of seeds is large as the total number of plans that may be produced scales up very fast. 5) There is no intuitive way to incorporate intra-cluster constraints during the clustering process.

5.2.2 Simulated Annealing

Simulated Annealing is a general purpose optimization procedure based on the thermodynamic process of annealing of metals by slow cooling. In the redistricting problem the goal is to draw a plan such that the user defined constraints, such as equal population, are satisfied. An example of an algorithm that applies simulated annealing to solve the problem of congressional redistricting is *Simulated Annealing Redistricting Algorithm (SARA)* (Macmillan, 2001). An outline of the algorithm is as follows:

1. Select an over-populated cluster as the donor cluster.
2. Choose, among the member polygons of the donor cluster, a polygon to be removed from the donor cluster.
3. If contiguity of the donor cluster (i.e., the connectedness of all its member polygons) would be lost by this removal, return to step 2.
4. Select a recipient cluster for the chosen polygon from amongst the neighboring clusters.
5. (a) If the transfer of the selected polygon from the donor cluster to the recipient cluster would decrease the combined population deviation of the donor and recipient clusters then accept it; (b) if the transfer would increase the combined population deviation of the donor and recipient clusters then accept it with a probability governed by the size of the deviation and the value of the temperature parameter.
6. If the transfer is accepted, calculate the new population deviations of the clusters and add one to the count of successful transfers.
7. If the aggregate population deviation of all clusters is within the target range then stop; otherwise if the numbers of successful and unsuccessful swaps have not been exceeded a threshold then go to Step 1; if the thresholds have been exceeded then reduce the value of the temperature parameter then go to Step 1.

While simulated annealing based methods perform better than informal or manual methods, they have several disadvantages. 1) An initial solution needs to be provided to the algorithm. 2) The final solution produced is therefore heavily dependent on the initial plan provided to the algorithm. 3) More than one spatial constraint cannot be easily incorporated in algorithms such as SARA. 4) There are no guarantees that a global optimum will be found.

5.2.3 Genetic Algorithms

Genetic algorithms are a subset of the evolutionary algorithms based on Darwin's theory of evolution. The basic idea is that each solution to the problem is coded as a bit string, taken to be a chromosome, possibly with a number of sub-strings that act as genes. At any given point in time, a number of such chromosomes are kept where each chromosome represents a solution to the problem. Natural selection is simulated by evaluating the fitness of each solution, measured by how well it solves the problem at hand, and giving the best individuals a higher probability of remaining in the solution pool during the next generation. To obtain new solutions, two operators are used – crossover and mutation. Crossover is implemented by combining bits of two different chromosomes to form a new solution. Mutation is implemented by randomly changing some bits or chromosomes. An application of genetic algorithms to zone designing is given is (Bacao, Lobo, & Painho, 2005). The algorithm takes as input a point representation of each polygon, and the number of zones or clusters (k). A polygon is represented using its centroid. The algorithm is defined as follows:

1. Generate a population of size p, where each population is a set of k points, according to the selected encoding. Thus each population forms a chromosome representing a possible solution.
2. Generate a plan for every chromosome within the population, by assigning each of the n polygon centroids to the closest centroid within the chromosome.
3. Evaluate the fitness of each plan, based on the chosen fitness function and contiguity check.
4. Apply selection, crossover and mutation operators, creating a new population.
5. Return to Step 2 until the stopping criterion is met.

Given enough time, a global optimum solution may be found by a genetic algorithm. However, to find a reasonable solution in a reasonable amount of time, care must be taken in encoding the solution space into chromosomes. The disadvantages of genetic algorithms are as follows: 1) They need many more function evaluations than linearized methods. 2) A lot of care needs to be taken while designing the encodings. 3) There is no guaranteed convergence even to a local minimum. 4) Finally, genetic algorithms cannot be applied to problems where the seeds are fixed and thus only one chromosome in the initial population pool.

5.2.4 Comparison with the CPSC family

The CPSC family of algorithms addresses several of the disadvantages of the approaches discussed in Sections 2.1, 2.2. and 2.3. The CPSC family does not require an initial input plan, i.e., an initial solution where *every* polygon is assigned to a district or cluster, to work upon, as is the case with SARA and the genetic algorithm for zone design. For example, for the congressional redistricting problem, SARA takes as input a set of districts with every polygon in the dataset assigned to a district, which are spatially contiguous but do not satisfy all the constraints such as the equal population constraint. SARA then improves upon this initial plan so that all the districts satisfy the constraint of equal population. The genetic algorithm for zone design also follows a similar approach where it begins with taking a set of input plans. Here again, each input plan is a possible solution to the congressional redistricting problem where each polygon in the dataset is assigned to a district. It then evaluates the fitness of each plan, based on the chosen fitness function and contiguity check, and applies selection, crossover and mutation operators until it finds a solution that meets the stopping criterion. The CPSC family, on the other hand, selects seeds from the dataset and then grows the clusters with the seeds as the starting points of the clusters. The seeds are simply single polygons selected as for growing clusters, and therefore do not constitute an input plan.

The CPSC family also defines a clear methodology to select seeds based on a pre-defined set of constraints, as opposed to the random selection of seeds by the graph partitioning algorithm. Most importantly, the CPSC family can incorporate any type of spatial or domain-specific constraints in the clustering process by the use of its heuristic function and other guidelines as defined in Section 5.3, and demonstrated in Section 5.4. There is no intuitive way to incorporate spatial constraints such as minimum distance between the seed and other polygons within the cluster within SARA and genetic algorithm. SARA randomly makes its move in order to avoid local optima; however, it has the risk of getting stuck at the local minima. As the CPSC family follows the A* search-like mechanism in order to grow the clusters, there is no risk of getting stuck at the local minima. Finally, CPSC can also easily be modified to work with fixed seeds, i.e. the seeds of the cluster cannot be changed or moved during the clustering process. A genetic algorithm will not work in this situation as the input population cannot be formed in this case.

5.3 Constrained Polygonal Spatial Clustering Algorithms

The main aim of our Constrained Polygonal Spatial Clustering (CPSC) algorithm is to grow clusters, satisfying constraints that can be used for spatial analysis and map formation. In order to facilitate the purposes of jurisdiction within a cluster that represents a district, the algorithm is designed to inherently produce spatially contiguous and compact clusters. This knowledge is embedded in the clustering process in the form of constraints. Towards this, we make use of the notions of *instance-level constraints*, and *cluster-level constraints*. A description of the different types of constraints in presented in Section 5.3.1.

Using the constraints mentioned above, the clusters are grown using an iterative search process. The underlying search algorithm used is A*-like search (Russell & Norvig, 2003). An outline of the A*-search algorithm has been presented in Section 5.3.1.

5.3.1 Preliminaries

A* Search Algorithm: A* is a best-first search algorithm that finds the least costly path from an initial node to the goal node. It uses a heuristic function ($F(n) = G(n) + H(n)$) that is a combination of a path cost function ($G(n)$) and an admissible distance function ($H(n)$) i.e. a distance function that does not overestimate the distance to the goal. The path cost function $G(n)$ measures the cost of arriving at the current node from the initial node, and the distance function $H(n)$ measures the estimated distance from the current node to the goal node.

Starting with the initial node, A* maintains a priority queue of nodes to be traversed, known as the open set, or OPEN. The lower $F(n)$ for a given node n is, the higher is its priority. At each step of the algorithm, the node with the lowest $F(n)$ value is removed from the OPEN queue and added to another queue known as the closed set, or CLOSED. The F and H values of its neighbors are updated accordingly, and these neighbors, which have not been already added to OPEN or CLOSED, are added to the OPEN queue. The algorithm continues until a goal node is discovered (or until the OPEN queue is empty). The F value of the goal is then the length of the shortest path. (Russell & Norvig, 2003). An outline of the algorithm is as follows:

1. Begin with the start node n_0.
2. Put n_0 on a queue called OPEN.
3. Create a queue called CLOSED that is initially empty.
4. If OPEN is empty, exit with failure.
5. Remove node n having the smallest $F(n)$ value from OPEN, and put it on CLOSED.
6. If n is a goal node, exit successfully.
7. Expand node n, generating the set M, of its neighbors.
8. Add the members of M not already on OPEN or CLOSED to OPEN.
9. Reorder the list OPEN in order of increasing F values.
10. Go to Step 5.

Spatial contiguity: A cluster of polygons is spatially contiguous when every polygon within the cluster shares at least a part of its boundary with at least one other polygon within the cluster. In other words, the number of connected components for a spatially contiguous cluster will always be 1.

Cluster compactness: Compactness is most commonly measured as an attribute of the shape of the cluster. A circle is the most compact shape for any cluster because it covers the most area within the smallest perimeter (Clayton, 2000). We define a compact cluster as a cluster that has a shape very close to that of a simple geometric shape and does not meander in space forming a snake or river like structure. Examples of simple geometric shapes are circle, rectangle and square. Different measures have been defined in order to compute the cluster compactness. For example - *radial compactness* measures the compactness of a cluster as the sum of Euclidean distances between the centroid of its polygons and the centroid of the cluster itself (d_{ij}). Thus $ss = \sum_{i \in C_j} d_{ij}$. The smaller the value of this index, the more compact the cluster is.

Different types of Constraints: In many cases there is some domain knowledge present. Instead of simply using this knowledge for validation purposes, it can also be used to "guide" or "adjust" the otherwise unsupervised clustering process (Grira, Crucianu, & Boujemaa, 2005). The resulting approach is known as the semi-supervised clustering or the process of constraint-based clustering (Basu, Banerjee, & Mooney, 2002). Constraint-based clustering makes use of the domain knowledge by transforming it into a set of constraints which are then applied during the process of grouping together the data objects being clustered. Constraints applied during the process of clustering can be of two types – instance-level constraints and cluster-level constraints.

Instance-level constraints are applied to the individual objects being clustered. There are two types of instance-level constraints, namely, must-link constraints and cannot-link constraints (Davidson & Ravi, 2005). The must-link constraints are the set of constraints that will be satisfied by the polygons that must belong to the same cluster. For an example of must-link constraints consider a group of spatially contiguous census tracts (say tract 10, tract 11, and tract 12) where the dominant population is that of a minority race. The constraint is that these census tracts must be clustered together. In order to implement this, must-link constraints will be applied to the census tracts 10, 11, and 12. For example, $must\text{-}link(tract\ 10, tract\ 11)$, $must\text{-}link(tract\ 11, tract12)$, and $must\text{-}link(tract\ 10, tract12)$.

Cannot-link constraints are the set of pair-wise constraints that will be satisfied by a pair of polygons if both the polygons are members of different clusters. On the other hand, the members of the same cluster will violate this set of constraints. For example, there may be a requirement that the census tracts across county boundaries cannot be clustered together. In other words, lets say that tract 1 belongs to county A and tract 2 belongs to county B. In this case, tract 1 and tract 2 cannot be clustered together. In order to implement this, cannot-link constraints will be applied to census tracts belonging to different counties. For example, $cannot\text{-}link(tract1, tract2)$.

Cluster-level constraints are applied to the cluster on the whole. Examples of cluster-level constraints are averaging or summation constraints (Davidson & Ravi, 2004). For example, in the formation of school districts, within a district each polygon must be at most *x* distance away from the school polygon. This will be categorized as a cluster-level constraint because it pertains to grouping “related” polygons into the same cluster. Another example is the constraint that specifies that each district must have a student population of *y* students. Other examples of cluster-level constraints include the constraints of spatial contiguity, and compactness

Heuristic Function based on Constraints: In order to incorporate the different types of constraints within the clustering process, the idea of a heuristic function (F), borrowed from heuristic search algorithms, is used. F is a combination of:

(1) A function that approximates the distance of the current state of the cluster to the goal state (H) thereby measuring the level of need of the cluster to grow further, and

(2) A cost function that measures the reduction in flexibility on the growth of the clusters (G).

Using the above, F is defined as a sum of the two, that is,

$$F = H + G \tag{1}$$

The distance function H takes into account the cluster-level constraints to find the distance of the current state of the cluster from its target state, and the cost function (G) looks at the effect of the growth of every cluster on the other clusters. Using this combination of H and G CPSC is able to make informed decisions about which cluster to grow, or which polygon to add to the selected cluster. The reduction in the flexibility of the growth of the clusters is viewed as a cost function because a choice based on H alone may have an adverse effect on the growth of the remaining clusters. With the addition of G to H we penalize a node if it restricts the growth of other clusters. In other words, we prevent CPSC from following a purely greedy approach.

In order to select the distance function (H) that approximates the distance of the current state of the cluster to the goal state, the first step is to identify which constraints are easily quantifiable, and which are the most important to satisfy. Using this information, the desired properties of the target clusters need to be identified. For example, while forming congressional districts, the most important constraint is equal population within a given margin of error, and spatial contiguity. Another important constraint is spatial compactness. Thus, we can formulate our target clusters to be spatially contiguous and compact clusters with equal population. Once the desired properties of the target clusters have been identified, in order to define H, we need to identify which constraints are cluster-level constraints. For example, among the most important constraints identified for the congressional redistricting problem, the constraints of equal population and spatial compactness are cluster-level constraints, while the spatial contiguity can be most easily translated into instance-level must-link and cannot-link constraints. Using the cluster-level constraints identified to arrive at the target or the goal state, the distance function H can be defined as:

$$H = d_1(Current\ C_j, Target\ C_j) + d_2(Current\ C_j, Target\ C_j) + \cdots + d_x(Current\ C_j, Target\ C_j) \tag{2}$$

where $d_x(Current\ C_j, Target\ C_j)$ is the distance between the current state of cluster C_j and the target state of cluster C_j based on cluster-level constraint x.

Thus, for the congressional redistricting problem, the distance function H will be defined as:

$$H = \frac{Req\ Pop - Current\ Pop}{Req\ Pop} + \frac{Target\ Compactness - Current\ Compactness}{Target\ Compactness} \quad (3)$$

Where $Req\ Pop$ is based on the domain knowledge available about the expected total population of each cluster produced, and this information can be easily used to describe the goal state. Furthermore, the constraint of cluster compactness dictates that the cluster must grow to form the most compact district. As stated before, a district with a circular shape would be the most compact, the compactness index of a circle measured using the Schwartzberg's index (Schwartzberg, 1996)(defined as the ratio of the square of the perimeter and the area) will always be 4π. Thus $Target\ Compactness = 4\pi$. The $Current\ Pop$, and the $Current\ Compactness$ are the measures of the current state of the cluster.

With the use of the cost function G our objective is to select a cluster to be grown that will preserve the maximum degree of flexibility for the other clusters to grow. In order to select a cost function that measures the reduction in flexibility on the growth of the clusters, we observe the effect of the growth of one cluster on the ability of growth of the other clusters. This function is mostly dictated by the domain-independent constraints of assigning every polygon to a cluster, and forming spatially contiguous and compact district. As example of a cost function G is as follows:

$$G = max_{i=0\ to\ k}\, max_{j=0\ to\ n}\left[(O(C_i) - O'(C_{i,j}))/(O(C_i))\right] \quad (4)$$

where k is the number of clusters, n is the number of polygons surrounding a cluster—i.e., neighbors—that have not yet been assigned to any cluster, $O(C_i)$ is the (outer) boundary of a cluster i (assuming all polygons within the cluster are contiguous) that is shared with polygons that are still not assigned to any cluster, and, $O'(C_{i,j})$ is the resulting new boundary of the cluster i after adding a new polygon j. Intuitively, this cost function says that if adding a new polygon makes it more compact – such as filling up a concave segment of the old boundary – then the cost will be negative (lowered); otherwise the cost will be positive as the cluster is growing more aggressively reducing the flexibility of the growth of other clusters which would benefit much more with the addition of new polygons. This cost function will promote a parallel cluster growth process.

Another example of a cost function G is as follows:

$$G = max_{i=0\ to\ k}\, max_{j=0\ to\ n}\left[(N(C_i\ ) - N'(C_{i,j}))/(N(C_i))\right] \qquad (5)$$

where k is the number of clusters, n is the number of polygons surrounding a cluster—i.e., neighbors—that have not yet been assigned to any cluster, $N(C_i)$ is the number of free neighbors of the cluster i and, $N'(C_{i,j})$ is the number of free neighbors of the cluster i after adding a new polygon j. Intuitively, this cost function will lead to a rush towards complete clustering, and will encourage one cluster to dominate the clustering process by rewarding a cluster for adding polygons that would give it more free neighbors. This cost function will promote a sequential cluster growth process.

In summary, please note that if H overestimates the distance of the current state of the cluster from the target state, the clustering process will not jump from one cluster to another. It will instead grow one cluster at a time. Therefore the clustering process will become sequential. On the hand, if H underestimates the distance then the clustering process will become considerably slower. Similarly, a more stringent cost function will result in a slower clustering process with every cluster selecting a polygon to grow very conservatively and vice-versa.

In Section 5.4 where we apply our algorithm to the congressional redistricting problem and the school district formation problem, spatial contiguity and cluster compactness are important properties of the desired target clusters; we chose the cost function based of the extent of the open boundary of the clusters defined in Equation 4, as this cost function penalizes the clusters the most if they are not spatially compact, and are instead distributed in space.

5.3.2 The CPSC Algorithm

The CPSC algorithm begins by selecting seeds from the dataset. As each seed will be grown to form a cluster, every seed represents a separate cluster. Because each seed represents an individual and different cluster, the seed selection follows a counter-intuitive path where every seed polygon must violate all *must-link constraints* w.r.t. to other seed polygons. Otherwise, the resulting seeds may be clustered within the same cluster, making the initial seeding invalid. Thus, the seeds are selected from the dataset using a systematic search based upon the available domain knowledge. This is done as follows: The heuristics based on the domain knowledge are measured for each polygon, for example, the pair-wise distance between the polygons, the population of each polygon, the area covered by each polygon, etc. Based on the desired properties of the target clusters, the most important constraint as identified by the domain experts is selected, and the corresponding property used in the constraint is implemented and computed for each polygon. For example, for the congressional re-districting problem, the constraint that every district should have equal population is considered the most important; therefore, the property to be computed is the population of each polygon. Next, the polygons are sorted in ascending order based on the computed property. Then we select the top k polygons in the sorted list that (1) violate the must-link constraints such as spatial contiguity, and (2) abide by any cannot-link constraints, where k is the pre-defined number of clusters to be detected.

Once the seeds are selected, the initial clusters come into existence, and a search process can begin. Adopting the A*-search algorithm, we assume that the initial clusters (consisting of the individual seeds) are the start state, and the target clusters are the goal state. Each cluster is then grown from the start state by adding polygons to the cluster one by one until the target cluster state is achieved. Adapting this search paradigm, at the beginning of every iteration the best cluster (BC) to be grown is selected. To achieve this, a heuristic function (F) is used (cf. Section 5.3.1). CPSC selects the cluster with the biggest need, that is the cluster with the largest F.

Upon the selection of the best cluster, the next step is to select the best polygon (BP) to be added to the best cluster. Toward this, first a set of potential polygons (PP) is selected that may be added to BC. This set consists of all previously unassigned, spatially contiguous neighboring polygons to BC, i.e. the polygons that share their boundary with BC ($SpatiallyContiguous(BC, p_i) = true$), and have not been assigned to any cluster so far. In case there are zero unassigned polygons remaining within the neighborhood of BC, then the neighboring polygons from the neighboring clusters, i.e., the clusters sharing some portion of their boundary with BC, are selected as potential polygons for BC. Every polygon within this set must abide by each intra-cluster constraint. A selection between them is then made on the basis of the heuristic function F'. F' is once again a combination of (1) a function (H') that approximates the distance of the current state of BC to the goal state after the addition of BP, and (2) a cost function (G') that measures the reduction in flexibility on the growth of BC after the addition of BP. Here CPSC selects the polygon that contributes most to the cluster, in other words, satisfies its need the most. Therefore, BP is the polygon that results in the smallest F for BC. This alternating strategy of selecting the cluster with the largest F as BC, and then selecting the polygon BP as the polygon that results in the smallest F for BC, allows every cluster to grow simultaneously, therefore giving every cluster the equal opportunity to select the best polygon for itself. If on the other hand, BC would be selected as the cluster with the smallest F, then the clusters would be forced to grow sequentially, and the property of compactness will be lost.

After BP is selected to be added to BC, it is necessary to check that by its addition the spatial contiguity of the clusters is still maintained. If BC and its neighboring clusters are spatially contiguous, the selected polygon (BP) is added to the best cluster (BC). This process goes on until:

1) All the polygons within the dataset have been assigned to a cluster, and the target state clusters are produced that satisfy the given set of constraints, OR

2) The algorithm enters the state of a deadlock, i.e. a set of clusters enters a cycle of repetitive states.

The condition "all the polygons have been assigned to a cluster" is not a constraint included in the instance-level or cluster-level constraints, as it is not domain dependent. Therefore, it is explicitly defined here so that the algorithm continues to grow the clusters until there is a polygon left that has not been assigned to any cluster. The condition "the target state clusters are produced" on the other hand is domain dependent, and refers to the pre-defined set of constraints, i.e. the algorithm continues to grow the clusters till there is a cluster that has not satisfied all the constraints. Both these conditions must be satisfied before the algorithm stops.

In the condition of "the algorithm enters the state of a deadlock" a cluster adds a polygon, then loses the polygon to another cluster, and then regains the same polygon over and over across successive iterations in a "tug-of-war" with another cluster. Formally, we define a set of clusters $(C_{1 \leq r < k})$ to be in a deadlock when at iteration *I* a cluster C_r is at state *x*, and at iteration *J*, where $J \leq I + k$, the cluster C_r is at state *x* again. The state of a cluster at any iteration *I* refers to the polygons that are the members of the cluster at iteration *I*. Figure 48 outlines the algorithm. The CPSC algorithm presented in Figure 48 can be applied to any domain given the dataset of polygons, the number of clusters, a set of constraints, and the heuristic function *F* based on the constraints.

CPSC Algorithm

Input: Dataset D of n polygons, Heuristic Function $F = G + H$, number of seeds k, Target state of the clusters, Set of intra-cluster constraints.

```
1.  Select k seeds $\{s_1, s_2, \dots, s_k\}$
2.  Initialize $k$ clusters by assigning each seed to a cluster $\{C_1 = s_1, C_2 = s_2, \dots, C_k = s_k\}$.
3.  While (There exists a polygon that has not been assigned to a cluster OR there exists a cluster that
    does not satisfy all cluster-level constraints OR there is a deadlock)
      i.    Select best cluster ($BC$) to grow.
      ii.   Find list of possible polygons ($PP$) as candidates for the growth of $BC$.
      iii.  Select best polygon ($BP$) from $PP$ to add to $BC$.
      iv.   Add $BP$ to $BC$ .
      v.    Update Cluster Status
      vi.   If updated, Continue
            Else (deadlock detected), Break.
    End While
```

Select k seeds

```
Select $k$ seeds $\{s_1, s_2, \dots, s_k\}$ such that:
  i.   The seed should be a polygon with a larger $F$ than
       the other non-seed polygons.
  ii.  Each seed polygon must violate the intra-cluster con-
       straints w.r.t. to other seed polygons.
Return $k$ seeds.
```

Find possible polygons

```
Select a set of neighboring polygons $pp$ such that
$pp = \{p_1, p_2, \dots, p_m\}$
  i.   Polygon $p_i$ such that $SpatiallyContiguous(BC, p_i) =$
       $true$.
  ii.  Polygon $p_i$ is a free polygon, i.e. $p_i$has not been as-
       signed to any cluster.
  iii. Polygon $p_i$ does not violate any intra-cluster con-
       straint.
If $pp = \emptyset$ , then select a set of neighboring polygons pn
such that:
  i.   Polygon $p_i$ such that $SpatiallyContiguous(BC, p_i) =$
       $true$.
  ii.  Polygon $p_i$ does not violate any intra-cluster con-
       straint.
  iii. Polygon $p_i$ was not added to $BC$ in the previous to
       previous iteration i.e. current iteration – 2.
Return $pp$
```

Update Cluster Status

```
If $BP$ was a free polygon, Increment iteration
        Return true
Else, Increment iteration
        Initiate deadlock watch
If deadlock detected, Return false
Else, Return true.
```

Select best cluster

```
  i.  Compute $F$ for each cluster.
      SelectThe CPSC* Algorithm

      Input: Dataset $D$ of $n$ polygons, $F$, $k$, Target
state of the clusters, Set of intra-cluster constraints.
```

Select best polygon

```
Compute the resulting $\forall p_i | p_i \in PP$: $F'(BC + p_i)$.
do
    Select $BP = p_i | min_i = 0\ to\ m\ (F'(BC + p_i))$
    If  $BP$ belongs to a neighboring cluster $C_j$
        If  $BP$ can be removed from $C_j$ without
        breaking its spatial contiguity, return true
        Else, return false
        If false,
            Set $BP = null$
            Remove $BP$ from $PP$
While($BP = null$)
Return $BP$
```

Deadlock Watch

```
Add $BC$ and current iteration to the deadlock
watch list.
If within the previous $k - 1$ items stored in the
deadlock watch list:
  If the iterations stored are in consecutive order
    If $BC$ is a member of $k - 1$ items,
        Return true
 Else,
  Return false.
```

Figure 48: The CPSC Algorithm

5.3.3 Extensions of CPSC

Though our algorithm guarantees completeness and optimality in the solution, it does not guarantee convergence because of the search process is cluster-centric instead of instance- or polygon-centric. First we define convergence as follows. An algorithm is said to converge when every polygon $p_i \in D$, where D is the complete dataset, is assigned to a cluster C_j i.e $\forall p_{1 \le i \ge n} \exists C_{1 \le j \ge k} | p_i \in C_j$. Currently CPSC does not guarantee convergence, i.e., every polygon may not be assigned to a cluster. Usually this will happen if the constraints provided by the user cannot be satisfied by the dataset.

If *the problem allows constraints to be softened or relaxed*, then, in order to guarantee convergence, we propose another algorithm known as **CPSC*** (Section 5.3.3.1). Furthermore, in order to improve the quality of the clusters obtained by CPSC*, we propose another extension of CPSC – **CPSC*-PS** (Section 5.3.3.2).

CPSC*. The CPSC* algorithm, presented in Figure 49, follows a similar approach to grow clusters as CPSC did. However, *CPSC* allows the users to relax their constraints to ensure that every polygon gets assigned to a cluster*. This relaxation of constraints is performed in two steps. First, CPSC* uses a weighted distance function H – thereby converting the hard cluster-level constraints to soft cluster-level constraints, and allowing the user to prioritize the constraints. And second, while selecting the potential polygon set to grow a cluster, CPSC* checks that all must-link constraints are met. However, if the best cluster (BC) has not achieved its target state yet, and there are no more polygons left that satisfy the desired constraints, then these constraints are relaxed, i.e. their margin is increased, so that the remaining unassigned polygons may become potential members of BC. For example, if there is a constraint that states that every polygon within the cluster must be at most 10 miles away from the seed polygon. However, there may exist polygons within the dataset that are more than 10 miles away from every seed. These polygons will never get assigned to a cluster. To overcome this situation the user may define the maximum distance allowed between any polygon within the cluster and the seed to be increased by 2 miles at a time. This condition will ensure that every polygon gets assigned to a cluster eventually no matter how far away they are from the seed polygons.

The weighted distance function *H* is defined as follows:

$$H = w_1 \times d_1(Current\ C_j, Target\ C_j) + w_2 \times d_2(Current\ C_j, Target\ C_j) + \cdots + w_x \times d_x(Current\ C_j, Target\ C_j) \quad (6)$$

where $d_x(Current\ C_j, Target\ C_j)$ is the distance between the current state of cluster C_j and the target state of cluster C_j based on cluster-level constraint x, and w_x is the weight assigned to cluster-level constraint x where $w_x > 0$ and $w_1 + w_2 + \cdots + w_x = 1$. The weights are assigned according to the priority of the constraints, and are user defined. The weighted distance function used by CPSC*, therefore, allows the user the flexibility to guide the growth of the clusters based on selected constraints, as opposed to the distance function used by CPSC that enforces every constraint equally on the clustering process. Furthermore, in the worst-case scenario, CPSC may lead to a situation where two or more clusters enter a deadlock. If this happens the algorithm will not converge. In order to avoid deadlocks; CPSC* initiates a *deadlock watch* as soon as a cluster adds a polygon that was previously assigned to another cluster. The deadlock watch stores the current state of the cluster. If across $k - 1$ consecutive iterations, the two or more clusters repeat the same state, a deadlock is detected. CPSC* then breaks the deadlock by forcing another cluster not involved in the deadlock to grow the properties of deadlock detection and breaking, and relaxing intra-cluster constraints when zero.

Theorem 1: *CPSC* guarantees convergence.*

Proof: Let us assume that there exists a polygon $p_i | p_i \notin C_j, 1 \leq j \geq k$, but CPSC* either (1) has stopped executing, or (2) has resided in a deadlock permanently. In case 1, assuming that CPSC* has stopped executing would imply that every polygons has been assigned to a cluster. This is

```
**The CPSC* Algorithm**
**Input**: Dataset $D$ of $n$ polygons, $F$, $k$, Target state of the clusters, Set of intra-cluster constraints.
  1. *Select k seeds*$\{s_1, s_2, \dots, s_k\}$
  2. **Initialize** $k$ clusters $\{C_1 = s_1, C_2 = s_2, \dots, C_k = s_k\}$, initialize iteration = 0.
  3. **While** (There exists a polygon that has not been assigned to a cluster)
       i.   *Select best cluster (BC)* to grow
       ii.  *Find list of possible polygons (PP)* to add to $BC$.
       iii. *Select best polygon (BP)* from $PP$ to add to $BC$.
       iv.  *Update Cluster Status*
       v.   **If** updated, Continue
            **Else** (deadlock detected)
                   **Select** the $BC$ from clusters not involved in a deadlock
                *Repeat the growth process for BC.*
     **End While**
```

```
*Find possible polygons*
**Select** a set of neighboring polygons $pn = \{p_1, p_2, \dots, p_m\}$
such that:
  - Polygon $p_i$ such that $Contiguous(BC, p_i) = true$ where
    $Contiguous(BC, p_i)$ is the *contiguity function* provided
    as input to the algorithm.
  - Polygon $p_i$ is a free polygon, i.e. $p_i$ has not been as-
    signed to any cluster.
  - Polygon $p_i$ does not violate any intra-cluster constraint.
**If** $pn = 0$ , then **Select** a set of neighboring polygons
$pn = \{p_1, p_2, \dots, p_m\}$ such that:
  - Polygon $p_i$ such that $Contiguous(BC, p_i) = true$ where
    $Contiguous(BC, p_i)$ is the *contiguity function* provided
    as input to the algorithm.
  - Polygon $p_i$ does not violate any intra-cluster constraint.
**If** $pn = 0$, then ***relax intra-cluster constraints***, and *repeat
process to select possible polygons*.
**Return** $pn$
```

```
*Select k seeds*
**Select** $k$ seeds $\{s_1, s_2, \dots, s_k\}$ such that:
  • The seed should be a polygon with a larger $F$ than the
    other non-seed polygons.
  • Each seed must be non-contiguous to each other based
    on the contiguity function.
**Return** $k$ seeds.
```

```
*Update Cluster Status*
**Add** $BP$ to $BC$.
**If** $BP$ was a free polygon, **Increment** iteration
        **Return** true
**Else, Increment** iteration
        **Initiate *deadlock watch***
**If** deadlock detected, **Return** false
**Else, Return** true.
```

```
*Select best cluster (BC)*
**Compute** $F$ for each cluster.
**Select** the best cluster $C_j = \max_{1 \le j \ge k} F(C_j)$, i.e., the
cluster with largest $F$.
**If** two or more clusters $(C_j, C_i)$ have the largest $F$ i.e.
$F(C_j) = F(C_i), C_j \neq C_i$
  **Select** the best cluster $C_j$ such that
  $nofreePolygons(C_j) > 0$ and
  $nofreePolygons(C_j) < nofreePolygons(C_i)$ where
  nofreePolygons$(C_j)$ is the number of free polygons
  spatially contiguous to cluster $C_j$.
**Return** the best cluster $C_j$.
```

```
*Select best polygon* (BP)
**Compute** the resulting $F'$  $\forall p_i | p_i \in pns$: $F'(BC + p_i)$
**Select** $p_i | \min_{0 \le i \ge q} F'(BC + p_i)$
**If** $BP$ belongs to a neighboring cluster $C_j$
  **If** $BP$ can be removed from $C_j$ without breaking its
  spatial contiguity, return true
  **Else**, return false
  **If** true, then $BP$ remains the same
  **Else**, remove $BP$ from $PP$
*Repeat the selection process of BP*
**Return** $BP$
```

```
*Deadlock Watch*
**Add** $BC$ and current iteration to the deadlock watch
list.
**If** within the previous $k - 1$ items stored in the
deadlock watch list:
  **If** the iterations stored are in consecutive order
    **If** $BC$ is a member of $k - 1$ items,
      **Return** true
**Else**,
  **Return** false.
```

Figure 49: CPSC* Algorithm

because CPSC* continues to grow all clusters until there exists a polygon that has not been assigned to a cluster. Since polygon p_i has not yet been assigned to a cluster, CPSC* will not stop executing. The only condition when p_i will not be directly taken up by it's neighboring cluster, let's say C_j, if some other cluster, C_y's F is greater than C_j's F. That is, $F(C_j) < F(C_y)$. Due to possible polygons are available, CPSC* guarantees that a polygon will be added to a cluster until there exists a free polygon within the entire dataset. Thus, the condition $F(C_j) > F(C_y)$ will become true. When this happens polygon p_i will be assigned to cluster C_j, and since every polygon has now been assigned to a cluster, CPSC* will stop executing. Thus, when CPSC* stops executing, every polygon will be assigned to a cluster.

In Case 2, assuming that CPSC* has resided in a deadlock permanently would imply that there does not exist a cluster that is not involved in the deadlock. This further means that none of the clusters have any free polygons that are contiguous to them. If this is the case, then all polygons would already be assigned to a cluster, and the algorithm would have converged already. Thus, CPSC* cannot reside in a deadlock permanently. Hence, our assumption has to be false for either case. Therefore, using proof by contradiction, we conclude that CPSC* guarantees convergence.

CPSC*-PS. In order to guarantee convergence CPSC* forces the hard constraints to be converted to soft constraints. To improve the quality of the results obtained by CPSC* such that the solution may come closer to satisfying the hard constraints, we propose an extension of CPSC* known as **CPSC*-PS**, where PS stands for Polygon Split. The assumption that CPSC*-PS makes is that the polygons can be divided into smaller polygons. Figure 50 presents an outline of the algorithm.

Once all the polygons have been assigned to a cluster, if the hard cluster-level constraints have not been satisfied, then CPSC*-PS selects a polygon from the cluster with the smallest F to be removed. This polygon is divided into two smaller polygons based on the underlying tessellation, and added to the dataset as unassigned polygons. CPSC*-PS then repeats the process of assigning these polygons to the clusters. This process is repeated until all cluster-level constraints have been satisfied, or until there exists a polygon that may be divided into smaller polygons.

```
The CPSC*- PS Algorithm
Input: Dataset $D$ of $n$ polygons, Set of intra-cluster constraints, $F=G+H$, $k$, Target state of the clusters
Select k seeds$\{s_1, s_2, \dots, s_k\}$
Initialize $k$ clusters $\{C_1 = s_1, C_2 = s_2, \dots, C_k = s_k\}$, initialize iteration = 0.
Loop: While (There exists a polygon that has not been assigned to a cluster)
  Select best cluster (BC) to grow
  Find list of possible polygons (PP) to add to BC.
  Select best polygon (BP) from PP to add to BC.
  Update Cluster Status
  If updated, Continue
  Else, deadlock detected
    Select the BC from clusters not involved in a deadlock
    Repeat the growth process for BC.
End While
If all cluster-level constraints have not been satisfied
  If there exists a polygon that can be divided into smaller polygons
    Select cluster $C_j$with smallest $F$
    Select polygon $p_i$ from $C_j$such that cluster $C_j - p_i$ does not violate the contiguity constraint
    Split polygon $p_i$ into $p_y$ and $p_z$ such that $p_y \cup p_z = p_i$
    Add $p_y$ and $p_z$ to dataset $D$.
    Go to loop
Else, end.
```

Figure 50: The CPSC*-PS Algorithm

Note: All the functions (for example, Select *k* seeds, etc.) for CPSC*-PS are the same as CPSC*, and therefore are not defined here again. Furthermore, as initially the algorithm uses CPSC* to produce clusters that are further improved upon using the polygon-split mechanism, and CPSC* guarantees convergence, CPSC*-PS also guarantees convergence.

5.4 Applications to Real-World Problems

In order to show the usefulness of our algorithm, we have applied CPSC and its extensions to two real world problems: (a) congressional redistricting and (b) formation of school districts. Both these problems can be interpreted as problems of cluster formation where each cluster represents a district. Each district or cluster is formed by grouping together polygons that follow certain constraints. Details of both the problems along with the constraints applied in both cases are described next.

Note that in our representation of the congressional redistricting problem, since all constraints involved are hard constraints, we use the CPSC algorithm. In the formation of school districts problem, there is a mixture of hard and soft constraints, and hence we use the CPSC* variant. As such, we also present the weights we use for the design of functions in the search process for the school districts problem.

5.4.1 The Congressional Redistricting Problem

Congressional redistricting has been a vexing problem for a long time. Once a state learns that it has been assigned k seats, it must divide its territory into k districts. This division is not an arithmetic division but a geometric one where there can be several ways of dividing the state territory into k districts (Hayes, 1996). This opportunity of being able to divide using several different methods leads to the phenomenon of political gerrymandering where any party could form districts for their own advantage.

The constraints that define a "good district" are as follows: (1) All the districts within a state should be equal in population, (2) Each district should be a single continuous territory, (3) Districts should be compact; Tentacles wriggling through the landscape are considered a bad design, (4) Districts should recognize the exiting communities of interest, (5) Districts should conform to existing natural and political boundaries when possible, and (6) Finally, under the US Voting Rights Act a district must not be drawn with the intent of excluding the minority candidates from election.

In case of any conflict among the above constraints, the highest priority is given to numerical equality and spatial contiguity. In our implementation we take into consideration only the first three constraints as they define the overall structure of the algorithm. Constraints 4, 5, and 6, are not incorporated due to lack of data. However, they can be applied as must-link and cannot-link constraints while selecting the possible set of polygons in step 3(c) of the algorithm.

Therefore, *the problem statement is to divide the geographic area of a state into k districts such that the total population within each district is nearly equal or within 1% margin of error. Each of these k districts must be spatially contiguous. Finally all of the k districts must be as compact as possible.*

Heuristics Used. The heuristic function F used by CPSC in order to determine the best cluster to grow, and the best polygon to add to the best cluster is defined based on the input dataset, and the constraints defined before the clustering process. For the congressional redistricting problem, the inputs to the algorithm are:

Dataset: Census Tracts of US as the set of polygons

Number of seeds: k

Target: k spatially contiguous and compact clusters $\{C_1, C_2, \ldots, C_k\}$ each containing population, x, with a margin of error of 1%.

Set of constraints: *Cluster-level Constraints*:

CS1. Each cluster must be spatially contiguous.

CS2. Each cluster must be compact.

CS3. Each cluster must contain equal population with a margin of error of 1%.

Instance-Level Constraints:

CS4. Set of spatial constraints as a set of must-link constraints between the census tracts.

CS5. Set of spatial constraints as a set of cannot-link constraints between the census tracts.

All the constraints mentioned above are hard constraints. Based on the above inputs, we define the heuristic function F as follows:

$$F = G + H,$$

where H, defined in Equation 3 in Section 5.3.1, measures the need for the respective cluster to grow further, and G, defined in Equation 4 in Section 5.3.1, is the cost of the reduction in the flexibility of the growth of the cluster

Thus, together with H, the best cluster (i.e. the best cluster that should be selected to grow) is one with the highest value of F, meaning one that is (1) furthest away from the target population and/or the least compact, and (2) the costliest to grow (akin to the min-conflict algorithm in conventional constraint satisfaction problems).

As alluded to earlier, we use the same rationale in designing the cost function G' for measuring the reduction in the flexibility of the growth of the best cluster while selecting the best polygon to add to the best cluster C_i as:

$$G' = \sum_{i=0}^{k}\left[(O(C_i\,) - O'(C_{i,j}))/(O(C_i))\right] \qquad (6)$$

To select the best polygon to add to a cluster, we select the neighbor that reduces the open boundary of the cluster the most, and takes the cluster closest to its target. Thus together with H', the best polygon will (1) increase the population of the cluster, (2) make it more compact, and (3) reduce the open boundary of the cluster.

Note that while we use a maximum function in Eq. (4), we use a summation function in Eq. (6). This is because when a free polygon (i.e. a polygon not yet assigned to any cluster) is added to a cluster, this action may considerably hinder the growth of another cluster. Therefore, we include the cumulative effect of the addition of a polygon to a cluster. Taken together, Eq. (4) allows us to pick the least costly cluster to grow, and Eq. (6) allows us to pick the least costly neighbor to add to that cluster.

Also, when computing F for identifying the seeds in the first place, since each "cluster" consists of only one polygon, the compactness measure is the same for each cluster (i.e.,= 1) and G is also the same for each cluster. Thus, in our application here, selecting the seeds from the dataset reduces the problem to sorting them by $F = H = x - current\ pop$ in descending order and selecting the top k polygons. Furthermore, since the seeds cannot be spatially contiguous we enforce a physical distance between them as follows: The physical distance between two seeds must be a function of r and k unless specified otherwise. That is, for seeds s_i and s_j the distance between them $dist(s_i, s_j) = f(r,k)$. where $r = \sqrt{\frac{Area\ of\ MBR}{k \times \pi}}$, $Area\ of\ MBR$ is the area of the enclosing minimum bounding rectangle of the dataset, and k is the number of seeds.

5.4.2 The School District Formation Problem

A school district is a geographic area in which the schools share a common administrative structure. A school district may have one or more public school. School districts are formed to ensure that no school is burdened with too many students, and that no student has to travel far to go to school. Therefore, each school district will approximately have a certain number of students, and every household will be within a certain distance from a school.

The formation of school districts is important because school districts hold great importance in the legislature of the community. The functioning of a school district can be a key influence and concern in local politics. A well run district with safe and clean schools, graduating enough students to good universities, can enhance the value of housing in its area, and thus increase the amount of tax revenue available to carry out its operations. Conversely, a poorly-run district may cause growth in the area to be far less than surrounding areas, or even a decline in population (Mann & Fowle, 1852).

Over the years due to the development of new businesses and new roads, populations have shifted and occupied new land. As a result, there are cases where a school district has lost the reason it even existed, or an existing school district is over-burdened with students and needs a new school.

The problem statement is therefore, to divide the geographic area of a state into districts such that each district has almost equal number of students, and every household must be within a threshold distance from the public school in the district.

Heuristics Used. For the school district formation problem, the inputs to the algorithm are:

Dataset: Census Blocks as the set of polygons

Number of seeds: k

Target: k spatially contiguous and compact clusters $\{C_1, C_2, \dots, C_k\}$ each containing population x, with a margin of error of 1%, and each polygon within a cluster must be within the threshold distance from the school polygon.

Similar constraints apply to this problem as the congressional districting problem. However, only spatial contiguity is a hard constraint. The equal population and compactness are soft constraints, because it is more necessary to assign every polygon to a cluster or school district in this case, rather than equal population and compactness. Other than these constraints, there is an added intra-cluster constraint of the threshold distance from the school polygon, i.e. every polygon within the school district must be no more than the threshold distance away from the polygon within which lies a school. This constraint is also a soft constraint, such that the threshold distance increased to guarantee convergence. Finally, the last constraint that applies to this problem is that the seeds will be fixed as the school polygons. This constraint is a hard constraint because the schools cannot be moved.

Based on the above inputs, we define the heuristic function *F* as follows:

$$F = G + H,$$

where $\quad = w_1 \times \frac{(Req\ Pop - Current\ Pop)}{Req\ Pop} + w_2 \times \frac{Target\ Compactness - Current\ Compactness}{Target\ compactness}$,

$w_1 + w_2 = 1$; G is the cost of the reduction in the flexibility of the growth of the cluster; $Required\ population$ is the total population divided by k. G is the same function as defined for the previous problem. Similarly, F' for selecting the best polygon is also the same. As the seeds are fixed for this problem, we do not need to execute the step to select seeds using F. The threshold distance between the polygons and the seed polygons is an intra-cluster constraint, and is therefore enforced when selecting potential polygons to be added to the best cluster as defined in Section 5.3 for the CPSC* algorithm. This distance is increased to allow larger distance between the polygons when relax constraints function is called by CPSC*.

5.5 Experimental Analysis

In this section we evaluate the CPSC algorithm suite by applying it to two well known redistricting problems - congressional redistricting as defined in Section 5.4.1 and the school district formation problem as defined in Section 5.4.2. We also compare our results for the congressional redistricting problem the results obtained by applying the graph partitioning, the simulated annealing algorithm (SARA), and the genetic algorithm for zone design described in Section 5.2. Furthermore, we examine the behavior of CPSC and CPSC* in these two experiments, and present CPSC*-PS to improve the quality of the clusters obtained by the CPSC* algorithm.

5.5.1 Evaluation of CPSC on the Congressional Redistricting Application

State of Nebraska. For this experiment, we used the census tract dataset for the state of Nebraska. The total number of polygons (census tracts) in Nebraska is 505. The state of Nebraska has been assigned 3 seats in the congress. Therefore the number of clusters (k) is equal to 3. The approximate population of each cluster or district must be equal to 570421 within a 1% margin of error. The 110th Congressional District Map for Nebraska is presented in Figure 51(f).

We first applied the graph partitioning algorithm presented in Section 5.2.1. Figure 51(a) shows the initial clusters formed based on a random run, and the final clusters produced in step 2 of the algorithm. Next, we applied SARA (presented in Section 5.2.2) on the same dataset. An initial plan needs to be presented to the algorithm as input. It then improves upon the clusters so that all the conditions are satisfied, and an optimum solution may be obtained. Figure 51(b) and Figure 51(c) present two random input plans and the respective results obtained upon the application of SARA. The input plans were selected with the following constraints: 1) Every input district is spatially contiguous. 2) Every input district is fairly compact to begin with. 3) Location of the districts is in the vicinity of the expected districts. 4) The input districts are designed such that no more than one initial seed selected by CPSC lies in any of the input district. A visual inspection of the input and the output plans show that the output plan is fairly dependent on the input plan. Furthermore, the algorithm does not promote the formation of compact districts.

The genetic algorithm for zone design, described in Section 5.2.2, was then applied to the Nebraska dataset. The results obtained are shown in Figure 51(d). Finally, the CPSC algorithm as described in Section 5.3 was applied to the Nebraska census tract dataset. As none of the constraints being considered for the congressional redistricting algorithm are difficult hard constraints, CPSC finds an optimal solution. The results obtained are presented in Figure 51(e).

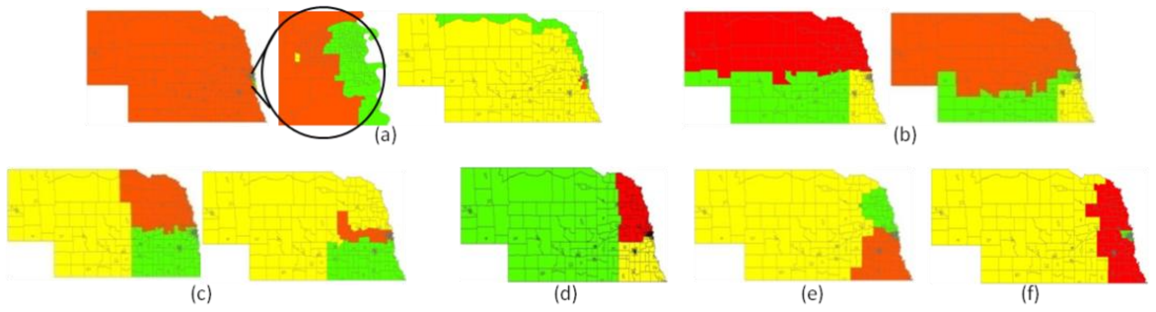


Figure 51: (a) Results of Graph Partitioning Algo. (b) & (c) Results of SARA: Input (left) and Output (right) plan 1 & 2 (d) Result of the Genetic Algorithm (e) Results of the CPSC Algorithm (f) 110th Congressional District Map for the state of Nebraska

Tables 12 and 13 present a comparison of the population distribution within the districts produced by all the methods listed above. We also compare the compactness of each district where the compactness is measured using the Schwartzberg Index (Schwartzberg, 1996). The desired population for each cluster is 570421. It is suggested that the actual population of each cluster or district must be within a 1% margin of error (M.O.E.). The margin of error is also computed for each district formed and listed in Tables 12 and 13. Please note that the spatial contiguity was implemented as a hard constraint in all the algorithms. Thus all the districts obtained by all the methods are spatially contiguous. From Tables 12 and 13 we can see that CPSC produces clusters/districts that fit the input criteria the best. All the districts have population within 1% margin of error, and the majority of districts are more compact than the districts obtained by SARA and the genetic algorithm.

Table 12: Comparison of clustering results for Nebraska Dataset

District	Graph Partitioning			Simulated Annealing (SARA)			Genetic Algorithm		
	Pop.	M.O.E	Compact.	Pop.	M.O.E.	Compact.	Pop.	M.O.E	Compact.
1	382209	-32.99	354.61	531708	-6.78	411.18	586140	2.76	92.81
2	75227	-86.81	3.34	586714	2.85	468.08	562373	-1.41	363.76
3	1253827	119.8	976.27	592814	3.93	95.95	562750	-1.34	142.44
Stdev	611426.45	107.18	492.69	33657.13	5.90	200.45	13614.36	2.39	144.26
Average	570421	0.00	444.74	570412	0.00	325.07	570421	0.00	199.67

Table 13: Comparison of clustering results for Nebraska Dataset (Contd.)

District	CPSC			Current Districts		
	Pop.	M.O.E	Compact.	Pop.	M.O.E	Compact.
1	573900	0.61	396.77	569318	-0.25	373.35
2	570408	0.00	134.46	574945	0.78	102.79
3	566955	-0.61	86.71	566590	-0.52	22.41
Stdev	3472.52	0.61	166.95	4260.50	0.69	183.86
Average	570421	0.00	205.98	570421	0.00	166.18

State of Indiana. In order to show the scalability of our algorithm, we apply our algorithm CPSC to a more complex dataset. For this purpose we use the census tract dataset of the state of Indiana. There are 1413 polygons (census tracts) in Indiana, and the number of seats assigned to Indiana is 9. Therefore, $k = 9$ with total expected population of each district equal to 675610. Figure 52 presents the districts formed for the Indiana dataset by the graph partitioning algorithm (Figure 52(a)), SARA (Figure 52(b)), the genetic algorithm for zone design (Figure 52(c)), CPSC (Figure 52(d)) and the 110th congressional district plan for the state of Indiana (Figure 52(e)). Tables 14 and 15 present a comparison between the results obtained by these four methods. Once again it is observed that CPSC produces districts that match the input criteria the most.

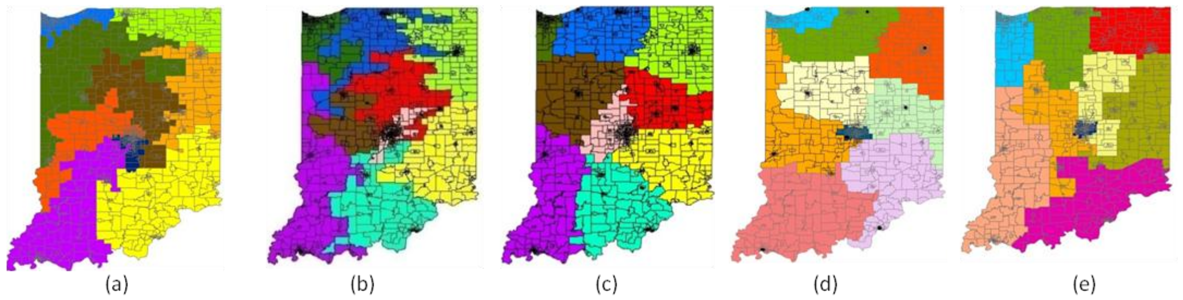


Figure 52: Results for the Indiana dataset (a) Graph Partitioning Result (b) SARA Result (c) GA Result (d) CPSC Results (e)Current Districts

Table 14: Comparison of clustering results for Indiana Dataset

	Graph Partitioning			Simulated Annealing (SARA)			Genetic Algorithm		
District	**Pop.**	**M.O.E**	**Compact.**	**Pop.**	**M.O.E**	**Compact.**	**Pop.**	**M.O.E**	**Compact.**
1	659901	-2.33	80.60	675669	0.01	132.84	396841	-41.26	105.52
2	660580	-2.22	49.63	681649	0.89	85.26	1029743	52.42	72.21
3	660517	-2.23	138.08	525207	-22.26	63.89	541885	-19.79	75.91
4	657017	-2.75	95.84	680292	0.69	77.69	776054	14.87	59.69
5	659339	-2.41	66.53	690360	2.18	68.18	735639	8.89	61.82
6	658714	-2.50	94.24	706184	4.53	105.33	698310	3.36	78.01
7	730879	8.18	102.05	700061	3.62	70.79	666736	-1.31	31.51
8	658926	-2.47	26.01	708589	4.88	83.209	602199	-10.87	61.18
9	734612	8.73	121.21	712474	5.46	41.5	633078	-6.3	35.65
Stdev	32424.23	4.80	34.83	57961.42	8.58	26.06	174101.04	25.77	22.41
Average	675609.44	0.00	86.02	675609	0.00	80.97	675609	0.00	64.61

Table 15: Comparison of clustering results for Indiana Dataset (Contd.)

	CPSC			Current Districts		
District	**Population**	**M.O.E**	**Compactness**	**Population**	**M.O.E**	**Compactness**
1	680395	0.71	15.11	677092	0.22	126.87
2	673922	0.05	50.03	672941	-0.4	92.29
3	673208	-0.06	101.45	675732	0.02	82.14
4	673206	-0.06	70.39	678656	0.45	64.14
5	675011	-0.09	55.47	657666	-2.65	13.99
6	672429	-0.46	67.42	693750	2.69	72.63
7	675097	-0.08	104.50	677947	0.35	68.42
8	673970	0.06	57.26	660700	-2.21	88.05
9	675179	-0.06	83.11	686001	1.54	63.96
St. dev.	1803.96	0.31	27.55	11210.50	1.66	30.04
Average	675609.44	0.00	67.19	675609.44	0.00	74.72

Finally, we also present a runtime comparison of the simulated annealing redistricting algorithm (SARA), the genetic algorithm for zone design, and constrained polygonal spatial clustering (CPSC) algorithm. The results are presented in Table 16. It can be observed that while CPSC takes more time than SARA when n is small, where n is the number of polygons being clustered, the time required by CPSC for a larger dataset does not scale up as fast as it does for the other two algorithms.

Table 16: Runtime Comparison (Minutes) on Intel Pentium processor T4300, 4GB memory

	Graph Partitioning	SARA	Genetic Algorithm	CPSC
N = 505	1440	1.9	65.52	5.85
N = 1413	2520	49.18	598.99	13.41

In both the experiments described above CPSC produces clusters that are spatially contiguous, compact, and conform to the other constraints presented to the algorithm as inputs. Furthermore in the congressional redistricting experiment a visual inspection of the Figures 51 & 52 show us that CPSC produces the most compact clusters, which is further verified by the compactness indices produced using the Schwartzberg index. The comparison of results in Tables 12, 13, 14 and 15 shows us that CPSC is the only algorithm that produces clusters with the most equitable population division within the districts. Furthermore Table 16 lists a runtime comparison of CPSC with the other three techniques. SARA produced the result the fastest (1.9 minutes) with a small dataset, however the plan produced was not optimal. CPSC produces a plan faster (13.41 minutes) than SARA (49.18 minutes) when the dataset size almost triples.

The main reason behind CPSC's superior performance is the use of heuristic function in seed selection, and in deciding which cluster to grow and which polygon to add to the selected cluster. This feature of parallel growth of all the clusters and unbiased selection of polygons is the novelty of CPSC and makes it better than other redistricting algorithms. Other than this, the holistic integration of constraints makes the resultant clusters a lot closer to the desired target. Finally, another unique feature of CPSC is the use of the cost function as a part of the heuristic function that measures the reduction in flexibility of clustering with every assignment of a polygon to a cluster. *Thus for redistricting purposes CPSC gives an optimal starting plan as opposed to randomized plans produced by other methods.*

The Graph partitioning algorithm runs in two phases. In the first phase it makes an initial guess where the partitions begin formation by a random seed selection. As a result the initial partition formed can be best described as a sub-optimal solution. Therefore, in order to get any meaningful results the bulk of the weight lies on the second phase where the initial partition formed is improved by an exchange of polygons within the clusters. Because of this large dependence on the initial plan every new run of the program is likely to produce a different plan. For a large dataset millions of plans may be produced which may make it very difficult for the user to select the best plan.

Simulated Annealing algorithms have become well known among optimization algorithms for they allow for a locally sub-optimal move in order to get out of local optima. However, there is no look-ahead property in the algorithm. Moreover, there is no space for incorporating explicit constraints in the algorithm. The results obtained show that there is a large dependence on the input plan provided to the algorithm. Therefore, it will be appropriate to use the simulated annealing approach to further improve a plan that is already very close to an optimal solution.

Genetic Algorithms have the ability to discover an optimal solution, but they may take very long time (cf. Table 16) before they are able to do so. Moreover, these algorithms are also heavily dependent on the input population, and the optimization function. The input population represents various possible solutions. Once again, the user has to be able to come up with good initial solutions to obtain a better final solution in a reasonable amount of time.

5.5.2 Evaluation of Extensions of CPSC on the School District Application

In the next experiment we used a partial census block dataset from the state of Texas to compare CPSC and CPSC*. Basically, we first assumed the constraints were hard when applying CPSC and then assumed that the same constraints could be relaxed when applying CPSC*. This does not imply that in the real district formation problem that constrains could be arbitrarily relaxed.

Our goal here was to highlight the impact that CPSC* could have on the redistricting problem if constraints were soft.

First, we randomly picked three blocks and designated them as school polygons, i.e. polygons with schools within and set $k = 3$. The dataset consists of 160 polygons and is shown in Figure 53(a). The problem statement for the school district formation problem has been described in Section 5.4.2. The expected result is to see k number of school districts, where k is the number of schools in the area. Each school district should have approximately equal number of students, and the farthest household in any district from the school must be within the threshold distance, i.e. the maximum distance allowed between a polygon and the school polygon. To begin with the desired student population within each district is 238452 with a margin of error of 1%, and the desired threshold distance is 10 miles. When CPSC was applied to this dataset, all the polygons were not assigned to a cluster because some of the polygons were further away from the school polygon. The result of CPSC is presented in Figure 53(b). However, as the problem statement dictates that the threshold distance may be relaxed, and thus may be treated as a soft constraint, we applied CPSC* next to this dataset. The threshold distance is increased by 5 miles. The result of CPSC* is presented in Figure 53(c). A visual inspection of CPSC* shows that every polygon has now been assigned to a cluster. Table 17 lists the population in each district, the margin of error of the population, and the compactness of each district formed by CPSC and CPSC*. It can be seen that for the districts obtained by CPSC*, none of them have a margin of error more than 1%.

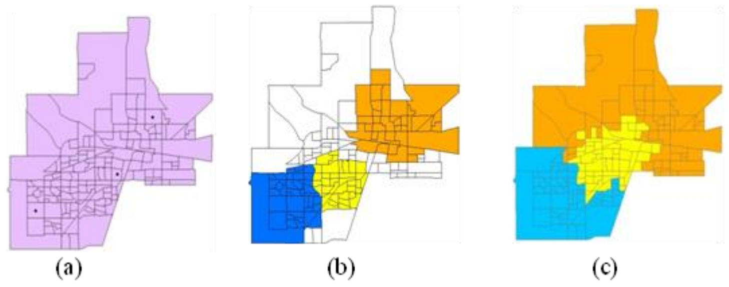


Figure 53: (a) School District dataset (b) CPSC Result (c) CPSC* Result

Table 17: School districts result statistics

	CPSC			CPSC*		
	Pop.	**M.O.E**	**Compact.**	**Pop.**	**M.O.E**	**Compact.**
1	165827	-30.4	66.93	238998	0.23	125.62
2	165888	-30.4	91.43	237053	-0.59	244.42
3	163086	-31.6	98.31	239305	0.36	193.04
St. dev.	1600.42	0.69	16.49	1221.25	0.52	59.58
Average	164933.67	-30.80	85.56	238452.00	0.00	187.69

For the school district experiment CPSC does not provide a solution for the problem, because an optimal solution does not exist within the dataset. However, if the problem is allowed to be modified such that the constraints can be relaxed, then CPSC* is able to provide an optimal solution for the school district problem.

In order to validate CPSC*-PS we conducted an experiment with a synthetic dataset that consists of a set of 20 polygons with 1000 population each (Figure 54(a)). The target is to divide the dataset into three clusters with a total population of 6666 each. When CPSC is applied to this dataset, the algorithm does not converge because the target can never be achieved. Once every cluster has achieved a population of 6000 each, all three of them are stuck fighting for the remaining two polygons. If on the other hand, CPSC* is applied to this dataset, and constraint of equal population is converted to a soft constraint of population between 6000 and 7000, the result obtained is three clusters with total population 6000, 7000 and 7000 respectively (Figure 54(b)). However, since we are still quite far from the initial target of 6666, we apply CPSC*-PS to this dataset. Each polygon within the dataset can be subdivided into two smaller polygons. The population gets divided equally within the two smaller polygons. CPSC*-PS when applied to this dataset results in three clusters with population 6500, 6500, and 7000 respectively (Figure 54(c)).

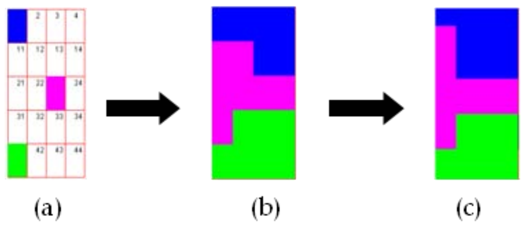


Figure 54: Application of CPSC* and CPSC*-PS on a synthetic dataset. (a) The synthetic dataset (b) Result of CPSC* (c) Result of CPSC*-PS

From these observations, we see the strengths and weaknesses of these three CPSC versions. In summary, CPSC is most suitable for situations where the constraints defined by the user are hard constraints, and a solution exists within the dataset. In case the constraints defined by the user are soft, and can be prioritized, CPSC* will be a better choice than CPSC. CPSC*-PS is the same as CPSC* with the additional steps of splitting polygons in order to optimize the clusters discovered with CPSC*. Thus CPSC*-PS is more computationally expensive than CPSC and CPSC*, and therefore must be used in situations where the polygons can be split into two or more smaller polygons such that the smaller polygons are still meaningful in the context of the application (e.g., splitting a county into census tracts while forming congressional districts within the state is meaningful because a census tract is a more compact polygon with smaller population, and can be easily divided into different congressional districts, but splitting a watershed into two is not meaningful since two watersheds belonging to two different rivers cannot be clustered together) and the result obtained by CPSC* is not sufficient.

5.5.3 Additional Analysis of CPSC Algorithms

Initial Seed Selection. In the section we further analyze the CPSC suite of algorithms. One would assume that the seed selection process has a great impact on the final results of the algorithm. To see the impact of the initial seed selection, we conducted an experiment with a small synthetic dataset. The dataset consists of a set of 27 polygons with 1000 population each. The target is to divide the dataset into three clusters with total population of 9000 such that each cluster is spatially contiguous and compact. To demonstrate the effect of seed selection, we modified the seed function to obtain different seeds. The results are presented in Figures 55(a, b & c). CPSC produces the same result irrespective of the initial seeds selected. Figure 55(c) further demonstrates that CPSC is robust enough to migrate the seeds from their original location such that the clusters satisfy all the user defined constraints when there is only one optimal solution within the dataset. However, in some cases this may not be the result, as shown in Figure 56. The figure demonstrates the different clusters produced for the Indiana census tract dataset based on the selection of different seeds. All the three plans meet the equal population criteria with 1% margin of error.

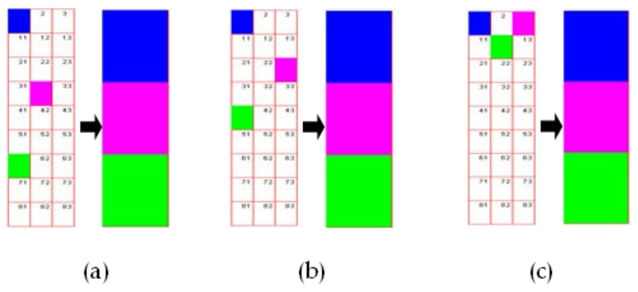


Figure 55: Application of CPSC on a synthetic dataset. Three initial seeds are color-coded as blue, pink, and green.

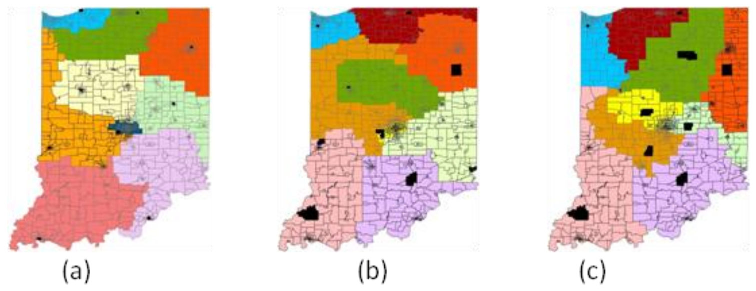


Figure 56: (a) CPSC results with minimum population seeds (b) CPSC results with maximum population seeds (c) CPSC results with maximum population seeds but with smaller distance.

5.6 Conclusion and Future Work

In this chapter we have proposed a new spatial clustering approach for polygon datasets instead of point datasets. This approach makes use of the available domain knowledge in the form of constraints that guide the clustering process. Our algorithm, called constrained polygonal spatial clustering (CPSC), views the clustering process as a search process, with seeds as the start states, and the desired clusters satisfying or optimizing the constraints as the goal states. As a result it can employ an A* search-like mechanism that allows CPSC to embed the constraints into the heuristic function that guides the "search" process. Specifically, CPSC strategically uses the set of constraints to select the initial seeds for the clusters, to compute the distance and cost functions to select the best cluster to grow next, and to select the best neighbor to add to the best cluster. We have demonstrated that CPSC is a complete and optimal algorithm. While CPSC works with hard constraints, we have developed two extensions of CPSC – namely, CPSC* and CPSC*-PS that work with both hard and soft constraints. These algorithms guarantee convergence. Thus, while redistricting is a NP-Complete problem (Bodin, 1973) we have successfully made the use of heuristic functions in order to achieve a feasible solution for this problem.

We have successfully applied the CPSC algorithm family to two difficult and important problems: congressional redistricting and school district formation. We have also shown that CPSC out performs other optimization approaches such as simulated annealing and genetic algorithms. There are several other such applications that can be greatly benefited by using CPSC redistricting algorithm. For example, electricity dispersion zones, traffic analysis zones, police precincts, etc.

In terms of future work, our immediate next step is to apply CPSC*-PS to a real application dataset, and perform further evaluations of the algorithm, along with developing a parameterized heuristic function that allows the user the flexibility to define a set of constraints, and along with providing their description, define the constraints as hard or soft. We will also be implementing the congressional redistricting problem more comprehensively by considering additional constraints such as the must-link constraint for minority-population areas, and test the scalability of our algorithm. In addition, we plan to consider other measures for compactness and testing with different cost functions, and see the difference in the clustering results. CPSC may further be benefitted by the use of the spatial characteristics such as topological relationships of the polygons. In particular, we intend to apply our framework to water resource management and drought mitigation making use of these additional features. There are also soft and hard constraints that are temporal (or seasonal) that we will need to consider.

Publications

This chapter appears in the following:

1. Joshi, D., Soh, L-. K., & Samal, A. (2009). Redistricting Using Heuristic-Based Polygonal Clustering. *IEEE International Conference on Data Mining*, (pp. 830-836). Miami, FL.
2. Joshi, D., Soh, L-. K. & Samal, A. (under review). Redistricting using Constrained Polygonal Clustering, submitted to *IEEE Transactions on Knowlegde and Data Engineering* in July 2010.

Chapter 6: Spatio-Temporal Polygonal Clustering with Space and Time as First-class Citizens

6.1 Introduction

The increasing numbers of tracking devices, sensors, and global positioning satellites have led to the generation of gigabytes of spatio-temporal data with relative ease for a variety of geo-physical variables. Detecting spatio-temporal clusters, i.e. clusters of objects similar to each other occurring together across space and time, has important real-world applications. For example, understanding the earth's atmosphere and the various geophysical processes that occur across time has been defined as a "Grand Challenge" (Stolorz, 1995). Some important applications include climate change analysis, drought analysis, detection of outbreak of epidemics (e.g. bird flu), bioterrorist attacks (e.g. anthrax release), and detection of increased military activity (Neill, Moore, Sabhnani, & Daniel, 2005). The detection and use of spatio-temporal clusters can enable us to discover trends and patterns that will in turn enable us to learn from the past, and be better prepared for the future (Aamodt, Samuelsen, & Skrondal, 2006).

Past research has focused on the discovery of spatio-temporal clusters by grouping objects with similar trajectories, detecting moving clusters, or discovering convoys of objects (Hwang, Chien-Ming Lee, & Lee, 2008), (Jeung, M.L. Yiu, Jensen, & Shen, 2008), (Jeung, Shen, & Zhou, Convoy queries in spatio-temporal databases, 2008), (Kalnis, Mamoulis, & Bakiras, 2005), (Yoon & Shahabi, 2009). All of this work is point-based and with the assumption that these points are mobile. Detecting spatio-temporal clusters of phenomena such as drought and disease outbreaks, however, is a fundamentally different problem as the geographic space is divided into a set of *polygons* (e.g. states, counties, etc.), and the polygons themselves do *not* move with the passage of time. However, a drought or disease may move across these fixed set of polygons spreading across several counties with the passage of time.

Therefore, the current techniques for detecting spatio-temporal clusters are inadequate for the aforementioned problems because of the following reasons:

- The current techniques are all point-based where only the longitude and the latitude of the object are used. While dealing with a polygonal dataset, if polygons, which are naturally rich objects with topological and structural properties, are represented as points, a significant amount of information is lost (Joshi, Samal, & Soh, A Dissimilarity Function for Clustering Geospatial Polygons, 2009a), (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b). For example, the length of shared boundary between two polygons is lost in point representation.
- These algorithms follow a time slicing approach; that is, snapshot clusters are formed at each time stamp and then a comparison is made between the clusters across various time-stamps to detect moving clusters. This approach translates to performing spatial clustering at each time stamp, and therefore it does not cluster entities that occur across different time stamps leading to an unbalanced treatment of space and time. While performing spatio-temporal clustering, time must be treated as a 'first-class citizen', i.e. time must be given equal importance as space. This is important to accurately track the dynamic clusters especially when clusters change significantly over time and space. An example is given in Section 6.2.3 to demonstrate the loss of information when giving more importance to the spatial dimension as compared to the temporal dimension.
- Convoys (Hwang, Chien-Ming Lee, & Lee, 2008), (Jeung, M.L. Yiu, Jensen, & Shen, 2008), (Yoon & Shahabi, 2009) are discovered in an object dataset where the objects move across space in time however their non-spatial attributes remain constant; for example, a convoy of vehicles moving along a highway. But not all objects move across space and time without changing their non-spatial attributes. For example, studying the movement of drought clusters, while the underlying polygons may remain constant, their non-

spatial attributes indicating the presence or absence of drought may change with time. Thus, the convoy detection algorithms would not work because the objects themselves do not move. Moreover, the attributes of the objects change.

In this chapter we present a spatio-temporal polygonal clustering algorithm known as the Spatio-Temporal Polygonal Clustering (STPC) algorithm. STPC is based on the density-based clustering principle as this clustering paradigm naturally adapts to concepts such as spatial auto-correlation and Tobler's first law of geography– 'All things are related, but nearby things are more related than distant things' (Tobler W. , 1979)(see Section 6.2.1 for details). STPC takes into account the spatial and topological properties of the polygons while taking into account the spatial neighborhood of the polygons as done previously in (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b). Furthermore, while the current density-based algorithms (e.g. DBSCAN (Ester, Kriegel, Sander, & Xu, 1996), P-DBSCAN (Joshi, Samal, & Soh, Density-Based Clustering of Polygons, 2009b)) only take into account the spatial neighborhood of the object being clustered at a particular time instant or time interval, we have re-defined the neighborhood of a polygon to not only take into consideration the spatial neighborhood of the polygon, but also the temporal neighborhood of a polygon. As a result of taking the spatio-temporal neighborhood of a polygon into account, we are able to treat both space and time as 'first-class citizens' – a feat that the other algorithms are not able to achieve. Thus, STPC is able to discover spatio-temporal polygonal clusters without detecting spatial clusters at each time slice. A unique property of our algorithm is that it has the ability to discover spatio-temporal clusters with holes in the spatial or temporal dimension.

We study the geospatial space that is divided into polygons. Thus the polygons themselves do not move in time, but their non-spatial attributes or properties may change with time. We examine the accuracy and efficiency of our algorithms in drought analysis, by discovering the spatio-temporal drought clusters in the state of Nebraska, in United States. Followed by which we discover the swine flu spread clusters within the state of California in United States, and finally we discover spatio-temporal crime clusters within the city of Lincoln, Nebraska. As a part of these experiments we show the effect of the different parameters of STPC, along with the robustness and scalability of our algorithm. Furthermore, while doing the drought analysis we compare and contrast the results of STPC with other spatio-temporal clustering algorithms presented in the literature and described in Section 6.2.2. The experimental results are presented in Section IV. We have shown that our algorithm outperforms other spatio-temporal clusters by retaining the maximum information about the clusters across space and time, and preventing one cluster from being split into two or more clusters. In other words, STPC is most capable of capturing big shifts within the spatio-temporal clusters, and maintaining the history of the cluster.

The rest of the chapter is organized as follows. Section 6.2 presents a brief introduction to the density-based principles as defined by (Ester, Kriegel, Sander, & Xu, 1996) and the related work to spatio-temporal clustering. Section 6.3 goes over the framework for our spatio-temporal neighborhood and the algorithm STPC. Section 6.4 presents our experimental results, and Section 6.5 gives our conclusion and future work.

6.2 Related Work

In this section we present a brief background on the principles of density-based clustering—on which our STPC algorithm is grounded—and the state of the art in discovering moving clusters. We also illustrate an example that demonstrates the difference in performing snapshot clustering and detecting convoys, versus treating both space and time as first class variables and detecting moving clusters.

6.2.1 Density-Based Clustering Principles

A density-based clustering algorithm hinges upon the assumption that a valid cluster must have sufficient density. As suggested by Tobler's first law of Geography (Tobler W. , 1979) and properties such as spatial autocorrelation (Zhang, Huang, Shekhar, & Kumar, 2003), phenomenon occurring in space and time naturally adapt to the density-based clustering paradigm. Ester et al. proposed a density-based clustering algorithm for point datasets, called DBSCAN. Here we list the main concepts of density-based clustering for points as defined in (Ester, Kriegel, Sander, & Xu, 1996). Let D be a database of points.

Definition 1: (ε-neighborhood of a point) The ε-neighborhood of a point p, denoted by $N_\varepsilon(p)$, is defined by $N_\varepsilon(p) = \{q \in D | dist(p,q) \leq \varepsilon\}$.

Definition 2: (directly density-reachable) A point p is directly density-reachable from a point q wrt. $(\varepsilon, MinPts)$ if (1) $p \in N_\varepsilon(q)$ and (2) $|N_\varepsilon(q)| \geq MinPts$ (core point condition).

Definition 3: (density-reachable) A point p is density reachable from a point q wrt. $(\varepsilon, MinPts)$ if there is a chain of points $p_1, \ldots, p_n, p_1 = q, p_2 = p$ such that for all i: p_{i+1} is directly density-reachable from p_i.

Definition 4: (density-connected) A point p is density connected to a point q wrt. ε, and if there is a point o such that both, p and q are density-reachable from o wrt. $(\varepsilon, MinPts)$, . Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive.

Definition 5: (cluster) A cluster C wrt. $(\varepsilon, MinPts)$ is a non-empty subset of D satisfying the following conditions:

1. $\forall\, p, q$: if $p \in C$ and q is density-reachable from p wrt. $(\varepsilon, MinPts)$ and $q \in C$. (Maximality)
2. $\forall p, q \in C$: p is density-connected to q wrt. $(\varepsilon, MinPts)$. (Connectivity)

Definition 6: (noise) Let $C_1, \ldots, C_k$ be the clusters of the database D wrt. parameters $(\varepsilon, MinPts)$, then we define the noise as the set of points in the database D that do not belong to any cluster C_i, i.e. $noise = \{p \in D | \forall i{:}\, p \notin C_i\}$.

The above density-based concepts are applied in the DBSCAN clustering algorithm that is used by the traditional moving cluster and convoy detection algorithms on point datasets. While applying the same principles of density-based clustering to polygons in the spatial and temporal dimensions, we, have extended the idea of the ε-neighborhood of a point that takes into account only the spatial neighborhood of the points to a spatio-temporal neighborhood of a polygon that takes into account both the spatial and temporal neighborhoods of polygons. By considering the temporal neighborhood of the polygon, we treat time as a 'first-class citizen' along with space (see Section 6.3).

6.2.2 Detecting Spatio-Temporal Clusters

Below we discuss the state-of-art in clustering algorithms detecting moving clusters in space and time. We discuss the Moving Cluster algorithm, the Coherent Moving Cluster algorithm, the Valid Convoy Discovery algorithm, and the Cluster Over Time algorithm. Followed by which we list the disadvantages of these algorithms, and explain how our approach is different.

The general approach followed by the well accepted *Moving Cluster* (MC) algorithm (Kalnis, Mamoulis, & Bakiras, 2005) is as follows: A density-based clustering (e.g., DBSCAN) is first performed on the moving objects at each timestamp to find snapshot density-connected clusters of arbitrary shapes and then the intersection snapshot clusters appearing during consecutive timestamps is detected as a moving cluster if they share at least a certain number of objects in common which is defined with respect to a threshold θ ($|c(t) \cap c(t+1)| \ / \ |c(t) \cup c(t+1)| \geq \theta$) where $c(t)$ and $c(t+1)$ denote two adjacent snapshot clusters at time t and $t+1$, respectively.

Jeung et al. (Jeung, M.L. Yiu, Jensen, & Shen, 2008) extended MC and proposed the *Coherent Moving Cluster* algorithm (CMC). CMC first performs density-based clustering at each timestamp to find snapshot clusters of arbitrary shapes. The following two conditions are then tested: first, a convoy must have clusters in at least k consecutive time stamps (lifetime constraint), and second, of the intersection of two consecutive snapshot clusters must have at least m objects in common. If both the conditions are true then it is detected as a convoy.

Yoon & Shahabi (Yoon & Shahabi, 2009) stated that the MC algorithms described above have a critical problem with accuracy – they tend to miss larger convoys and retrieve invalid ones where the density-connectivity among the objects is not completely satisfied. To overcome this problem, the authors proposed the algorithm VCoDA (*Valid Convoy Discovery Algorithm*). This algorithm works in two phases: first a set of all partially connected convoys is discovered from a given set of moving objects using the PCCD algorithm (Yoon & Shahabi, 2009), and then the density-connectivity of each partially connected convoy is validated using the DCVal algorithm (Yoon & Shahabi, 2009) to obtain a complete set of valid convoys. The first phase of VCoDA extends the CMC algorithm by scanning through the entire time span, and updating a set of density connected snapshot clusters incrementally by consecutive ones with sufficient objects in common under four operations (i.e., insert, extend, delete, and return). This approach is further extended in the second phase such that the density-connectivity of each partially connected convoy is incrementally verified at every timestamp either by immediate, single re-clustering, or recursive validation.

In (Lai & Nguyen, 2004) a simple set of formulas was proposed to predict which paired objects will move in the ε-neighborhood of each other. From these pair-wise ε-neighborhood relationships, a COOT (Core Object Over Time) algorithm is constructed to identify which objects will become core objects of future density-based clusters. This information reveals where, when, and how long the dense concentrations of objects may happen. Contents of density-based clusters over time can also be constructed using the Clusters Over Time (COT) algorithm with higher space and computation cost.

The first three algorithms mentioned above apply a time-slice approach i.e. they first perform spatial clustering using DBSCAN at each time stamp in order to discover spatial clusters.

These are then compared consecutively in the presence of some user defined constraints, and spatio-temporal clusters or moving clusters are detected. While VCoDA is an improvement over the other moving cluster discovery algorithms (MC and CMC), as this algorithm discovers the maximum number of valid convoys or moving clusters, some post-processing of the results is required to prune out the invalid convoys. The authors in (Lai & Nguyen, 2004) claim to move away from the time-slice approach by detecting core objects over time. However, their algorithm – COT – fails to do so completely. This is because they compare the core objects previously detected across small time intervals to detect spatio-temporal clusters across that particular time interval. Thus one may argue that the algorithm detects several incomplete clusters, i.e. several small clusters which should in fact be a part of the same cluster are detected. Moreover all the above described algorithms assume that the dataset consists of objects with similar non-spatial attribute values. They cannot distinguish between objects that lie within the ε-neighborhood of each other but are not similar to each based on their non-spatial attributes. While this may seem trivial, however, producing clusters using the density-based scheme that have uniform non-spatial attributes cannot be simply implemented using a pair-wise distance measurement methodology. This is because while density-connectivity is a transitive property flowing within the cluster from one object to the next, similarity between objects based on non-spatial attributes is not a transitive property. Finally all the above mentioned techniques are all point-based, and cannot be directly extended to polygons.

We adopt the idea of producing spatio-temporal clusters using the density-based clustering methodology in our approach as well. However, instead of performing spatial clustering across each time interval, we follow an approach that allows us to detect clusters spanning across the spatial and temporal dimensions simultaneously. We also incorporate a strategy that allows us to detect clusters with similar non-spatial attributes. Finally our approach is designed to cluster polygons instead of points, even though it can be easily modified to be implemented for point datasets as well. In Section 6.4 we present a comparison of these point-based algorithms with our polygon – based algorithm STPC.

6.2.3 An Example

Consider the polygons as shown in Figure 57. The orange polygons represent the counties with drought at each time stamp. The centroids of the counties are marked as dots.

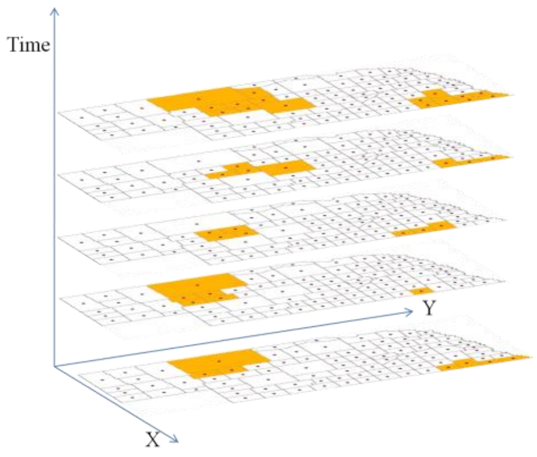
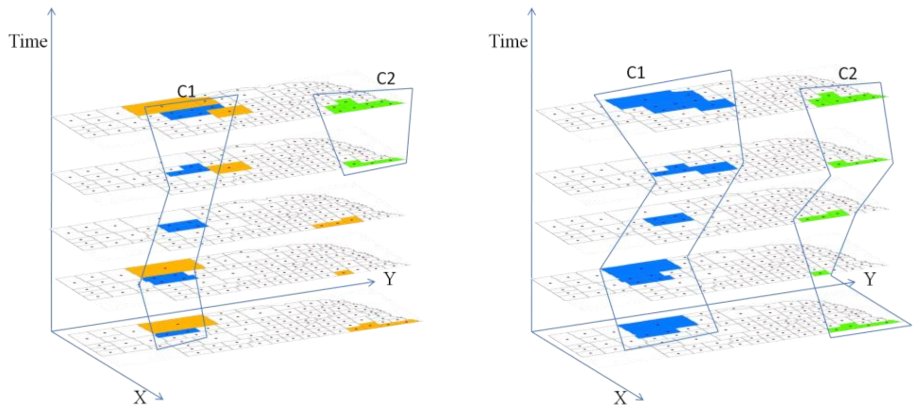


Figure 57: Sample dataset of polygons with drought at each time stamp . The centroids are shown as dots within each polygon.

Traditional trajectory clustering algorithms discussed on the previous sub-section perform snapshot density-based clustering (using algorithms such as DBSCAN (Ester, Kriegel, Sander, & Xu, 1996)) at each timestamp. Due to the snapshot clustering approach, and constraints such as the minimum number of objects (m) – i.e. the minimum number of common points that must be present within the two consecutive clusters and the lifetime constraint (k) – i.e. the minimum number of time stamps across which the cluster must exist, sudden changes that may happen in the cluster are not captured as part of the evolving cluster and instead viewed as the stop points of the cluster. As a result, the cluster breaks into multiple clusters leading to loss of information about the structure and contents of the cluster. For example, for the drought polygons shown in Figure 57, if we set m = 2 (i.e., the minimum number of objects), and k = 2 (i.e., the lifetime constraint), two clusters are detected as shown in Figure 58(a). The orange polygons in

Figure 58(a) are the drought polygons that do not get included in any cluster due to the constraints of minimum number of points that must be common within each sub-cluster formed at each time stamp and the minimum lifetime constraint. Thus, it can be seen that information may be lost using the techniques described in the previous section.

Our approach, on the other hand, gives time equal importance as space, and performs spatio-temporal clustering across time and space concurrently. As a result, two spatio-temporal clusters are detected (Figure 58(b)). In a way, our approach considers not just *n* polygons, i.e. the number of polygons at any timestamp, but $n \times t$ (i.e., *n* polygons multiplied with *t* time stamps) objects, and clusters them together without any bias. As a result no information is lost as compared to the traditional approach. For example, the cluster (C2) shown in Figure 58(a) becomes a part of a bigger cluster – cluster C2 in Figure 58(b). This information was not captured by the point-based approaches because there were none/or only one common points within the snap-shot clusters at time stamp t1, t2 and time stamp t4. However, because of the use of a spatio-temporal neighborhood as described in Section 6.3, the complete cluster is detected using our proposed approach. Furthermore, the second cluster (C1) detected by our approach is also more complete as compared to the cluster C1 shown in Figure 58(a). This is because of considering the polygons spatial and topological properties into account that were lost in the point representation.



(a) (b)

Figure 58: (a) Point-based spatio-temporal clusters formed using snapshot clustering approach. (b) Polygonal spatio-temporal clusters using time as a first-class citizen.

6.3 Spatio-Temporal Polygonal Clustering

Consider a set of polygons that exist in space and time. Each polygon has three categories of attributes associated with them. The first category is the space 'where' the polygon exists and is referred to by a set of spatial attributes, the second category is the time 'when' the polygon exists and is referred to by a set of temporal attributes, and the third category is 'what' the polygon is, and is referred to by a set of non-spatial, non-temporal attributes. A polygon that can be represented using all the above three categories of attributes is known as a spatio-temporal polygon.

In the following section we present the density-based concepts for spatio-temporal polygons that are based on the density-based concepts for points presented by Ester et al in (Ester, Kriegel, Sander, & Xu, 1996), followed by our algorithm STPC that is used to detect spatio-temporal clusters. Furthermore, each spatio-temporal cluster is dynamic in nature, i.e. each may have a different lifetime, and each cluster may spread across space with the passing of time differently. In order to capture these movements of the spatio-temporal clusters discovered by STPC, we first define the various types of movements possible for a spatio-temporal polygonal cluster, and then present the DMSTC algorithm that discovers the movements of a spatio-temporal cluster that it has undergone in its lifetime.

6.3.1 Density-Based Concepts for Spatio-Temporal Polygons

Given below are the definitions for the density-based concepts for spatio-temporal polygons.

Definition 1: A *spatio-temporal polygon* p_s^t is a polygon that exists at the location indexed by s and at the time interval indexed by t. (Henceforth, all polygons are spatio-temporal polygons unless specified otherwise.)

Definition 2A: A *spatial neighbor* of polygon p_s^t is any polygon p_k^t such that $dist(p_s^t, p_k^t) \leq \varepsilon$ where $dist(p_s^t, p_k^t)$ is any distance function that computes the physical distance between two polygons. Note that the temporal aspect is constant or reduced to a fixed interval or time instant in this case.

$SNeighbor(p_i^{t'}, p_j^{t''}|\varepsilon)$ is true iff $dist(p_i^{t'}, p_j^{t''}) < \varepsilon$ and $t' = t''$.

Definition 2B: The *spatial neighborhood* of a polygon p_s^t given ε, $SN(p_s^t|\varepsilon)$, is the set of the spatial neighbors of the polygon, that is $p_j^{t''} \in SN(p_s^t|\varepsilon)\ if\ SNeighbor(p_s^t, p_j^{t''}|\varepsilon)$.

Definition 3A: A *temporal neighbor* of a polygon p_s^t is a polygon that occupies at least some of the space indexed by s, $\varphi(p_s^t)$, at the time intervals $t \pm \omega$ where ω is a user-defined parameter to define the extent of the temporal neighborhood of a polygon. Note here, in contrast to the spatial neighborhood, the spatial dimension is instead held to a constant space.

$TNeighbor(p_i^{t'}, p_j^{t''}|\omega)$ is true iff $\varphi(p_i^{t'}) \cap \varphi(p_j^{t''}) \neq \emptyset$ and $|t' - t''| \leq \omega$.

where $\varphi(p_i^{t'})$ refers to the area covered by the polygon $p_i^{t'}$ at time instant t'.

For example, for a static set of polygons such as geospatial polygons, if $\omega = 3$, the temporal neighbors of polygon p_i^t is the set of spatio-temporal polygons $TNeighbors(p_i^t) = \{p_i^{t-3}, p_i^{t-2}, p_i^{t-1}, p_i^t, p_i^{t+1}, p_i^{t+2}, p_i^{t+3}\}$.

Definition 3B: The *temporal neighborhood* of a polygon p_s^t given ω, $TN(p_s^t|\omega)$, is the set of the temporal neighbors of the polygon, that is $p_j^{t''} \in TN(p_s^t|\omega)\ if\ TNeighbor(p_s^t, p_j^{t''}|\omega)$.

Definition 4: The *spatio-temporal neighborhood* of polygon p_s^t given ε and ω, $STN(p_s^t|\varepsilon, \omega)$, is the union of the spatial neighborhood and the temporal neighborhood .

$$STN(p_s^t|\varepsilon, \omega) = SN(p_s^t|\varepsilon) \cup TN(p_s^t|\omega)$$

Figure 59 shows the spatio-temporal neighborhood of polygon p_s^{t2}. The red polygon is the polygon p_s^{t2}, and the green polygons form the spatio-temporal neighborhood of the polygon p_s^{t2}, taking $\omega = 1$ and $\varepsilon = 1$.

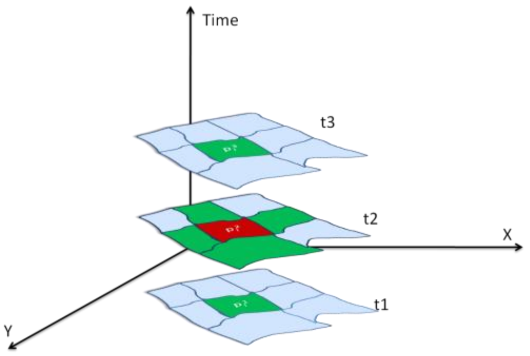


Figure 59: Spatio-temporal neighborhood (green polygons) of polygon $\mathbf{p_i^{t2}}$ (red polygon)

Definition 5: *Core spatio-temporal polygon* is a polygon that has at least *MinPoly* number of polygons in its spatio-temporal neighborhood i.e $|STN(p_s^t|\varepsilon,\omega)| \geq MinPoly$.

Definition 6: A spatio-temporal (ST) polygon $p_j^{t'}$ is *directly density-reachable* from another spatio-temporal polygon $p_i^{t''}$wrt. $(\varepsilon,\omega,MinPoly,MinS)$ if (1) $p_j^{t'} \in STN\left(p_i^{t''}|\varepsilon,\omega\right)$, (2) $|STN\left(p_i^{t''}|\varepsilon,\omega\right)| \geq MinPoly$.

Definition 7: A polygon $p_j^{t'}$ is *density reachable* from another polygon $p_i^{t''}$ wrt. $(\varepsilon,\omega,MinPoly,)$ if there is a chain of ST-polygons $p_1^{t'},\ldots,p_n^{t''},p_1^{t'} = p_j^{t'},p_n^{t''} = p_i^{t''}$ such that for all i: $p_{i+1}^{t'}$ is directly density-reachable from $p_i^{t'}$.

Definition 8: A polygon $p_j^{t'}$ is *density connected* to another polygon $p_i^{t''}$ wrt. $(\varepsilon,\omega,MinPoly)$ and if there is a polygon p_k^t such that both, $p_j^{t'}$ and $p_i^{t''}$are density-reachable from p_k^t wrt. $(\varepsilon,\omega,MinPoly)$. Density-connectivity is a symmetric relation. For density reachable polygons, the relation of density-connectivity is also reflexive.

Definition 9: A *spatio-temporal cluster* C wrt. $(\varepsilon,\omega,MinPoly)$ is a non-empty subset of P satisfying the following conditions:

1. $\forall\, p_j^{t'},p_i^{t''}$: if $p_j^{t'} \in C$ and $p_i^{t''}$ is density-reachable from $p_j^{t'}$ wrt. $(\varepsilon,\omega,MinPoly)$ then $p_i^{t''} \in C$. (Maximality)

2. $\forall p_j^{t'}, p_i^{t''} \in C$: $p_j^{t'}$ is density-connected to $p_i^{t''}$ wrt. $(\varepsilon, \omega, MinPoly)$. (Connectivity)

3. $\forall p_j^{t'}, p_i^{t''} \in C$, $d_N\left(p_j^{t'}, p_i^{t''}\right) = 0$. (Strong Uniformity) or

   $\forall p_j^{t'}, p_i^{t''} \in C$, $d_N\left(p_j^{t'}, p_i^{t''}\right) \leq \alpha$. (Weak Uniformity)

where $d_N(p_i^t, p_j^t)$ is the distance between the non-spatial attributes of the polygons and is computed using the Euclidean distance function, and α is a user-defined input parameter.

Definition 10: Let $C_1, \ldots, C_k$ be the clusters of the database P wrt. parameters $(\varepsilon, \omega, MinPoly)$, then we define the *outliers* as the set of polygons in the database P that do not belong to any cluster C_i, i.e. $outlier = \{p_i^t \in D | \forall i, t{:}\, p_i^t \notin C_i\}$.

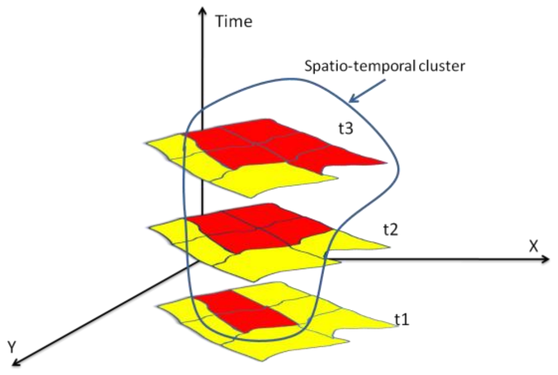


Figure 60: A Drought Spatio-Temporal Cluster (red polygons)

6.3.2 Spatio-Temporal Polygonal Clustering (STPC) Algorithm

In order to detect spatio-temporal clusters, we propose a new algorithm called Spatio-Temporal Polygonal Clustering (STPC) algorithm. Unlike other algorithms as defined in Section 6.2.3, STPC does not perform density-based clustering at each time stamp. STPC (presented in Figure 61), instead, uses the density-based concepts defined above for spatio-temporal polygons in order to detect a spatio-temporal cluster that extends both in space and over time, thus treating time as a first-class citizen along with space, and removing the need to find the intersection of snapshot clusters across consecutive time stamps.

Given a dataset of spatio-temporal polygons (P) STPC begins with a polygon (p_s^t) that has not been assigned to any cluster previously. If p_s^t meets the necessary conditions to be defined as a core spatio-temporal polygon, it is assigned a new cluster CID. It then calls the *expandST-Cluster* method that examines each polygon that falls in the spatio-temporal neighborhood of p_s^t based on $\omega\ and\ \varepsilon$. If a spatio-temporal neighbor has similar non-spatial attributes to p_s^t, and to every other spatio-temporal polygon that has already been assigned to cluster CID, i.e. if $\left(d_N\left(p_s^t, p_j^{t'}\right) = 0\right)$, the neighbor is assigned to the same cluster as p_s^t, and the *expandST-Cluster* method is subsequently called recursively on this neighbor. This process is repeated as long as there exists a polygon that has not been assigned to a cluster, or classified as an outlier.

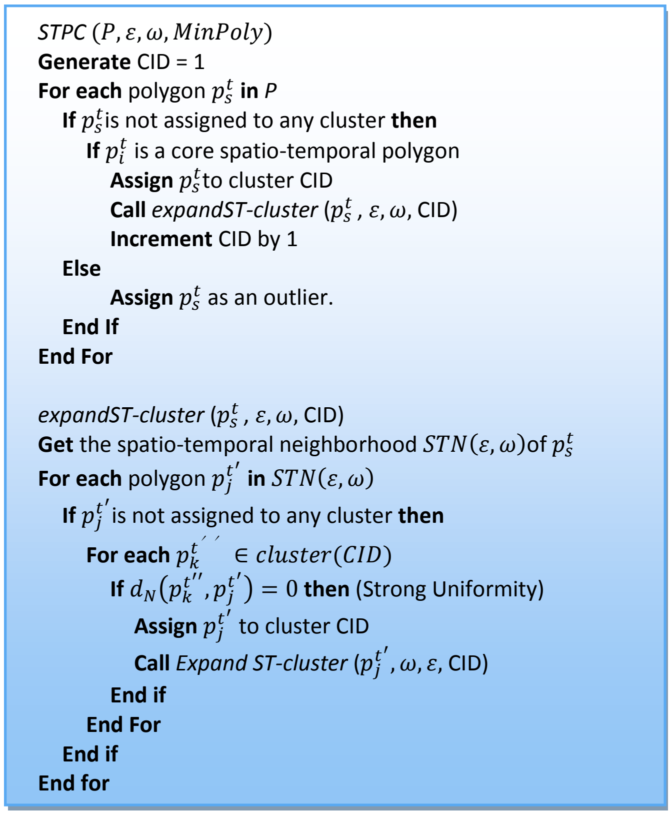
It should be noted that the test $d_N\left(p_s^t, p_j^{t'}\right) = 0$ is valid only in the case of categorical non-spatial attributes and the case reduces to a strong uniformity one For example, if the polygons are classified as "drought" or "no-drought" polygons, and $d_N\left(p_s^t, p_j^{t'}\right) = 0$; i.e., if either both the polygons p_i^t and $p_j^{t'}$ are classified as drought polygons, or they both are classified as no-drought polygons, only then both the polygons will be assigned to the same cluster. In order to handle non-spatial attributes having continuous values or ordinal values, the test $d_N\left(p_s^t, p_j^{t'}\right) \leq \alpha$ can be used instead (for weak uniformity), where α is a user-defined parameter.

Since our algorithm follows the structure of the density-based clustering algorithm DBSCAN, the time complexity of our algorithm is the same as DBSCAN which is $O(n^2)$ without the use of an indexing structure, where n is the number of data points. If an indexing structure such as a R* tree is used, then the time complexity will be reduced to $O(n\log n)$. Looking more closely, we find that the time complexity of our algorithm can be re-represented as $O(t \cdot n \log (t \cdot n))$, where t denotes the number of time stamps, whereas the time complexity for the conventional piecemeal approach is $O(t\cdot(n \log n))$. In most applications, $t \ll n$; hence both the algorithms will have a time complexity of $O(n \log n)$. However, STPC will typically run slower.

6.3.3 Selecting Input Parameters

In order to select the appropriate ε for a polygonal dataset, one of the following strategies may be followed:

1. Using a distance function such as the Hausdorff distance function, compute the pair-wise distance between the polygons within the dataset. Based on the set of the pair-wise distances, ε may be selected as the: mode of the set, the median distance value within the set, or the average distance. However, please note that a bigger ε value may result in the aggregation of two or more clusters within the same cluster.



```
STPC $(P, \varepsilon, \omega, MinPoly)$
Generate CID = 1
For each polygon $p_s^t$ in $P$
    If $p_s^t$ is not assigned to any cluster then
        If $p_i^t$ is a core spatio-temporal polygon
            Assign $p_s^t$ to cluster CID
            Call expandST-cluster ($p_s^t$, $\varepsilon, \omega$, CID)
            Increment CID by 1
    Else
            Assign $p_s^t$ as an outlier.
    End If
End For

expandST-cluster ($p_s^t$, $\varepsilon, \omega$, CID)
Get the spatio-temporal neighborhood $STN(\varepsilon, \omega)$of $p_s^t$
For each polygon $p_j^{t'}$ in $STN(\varepsilon, \omega)$
    If $p_j^{t'}$ is not assigned to any cluster then
        For each $p_k^{t''}$ $\in cluster(CID)$
            If $d_N\left(p_k^{t''}, p_j^{t'}\right) = 0$ then (Strong Uniformity)
                Assign $p_j^{t'}$ to cluster CID
                Call Expand ST-cluster ($p_j^{t'}$, $\omega, \varepsilon$, CID)
            End if
        End For
    End if
End for
```

Figure 61: The Spatio-Temporal Polygonal Clustering (STPC) Algorithm with Strong Uniformity.

2. Based on the knowledge of the dataset, the user may identify two polygons that must never be clustered together. Compute the geographic distance between these polygons, and then set ε to a value smaller than the resulting distance between the two polygons.

In order to select the appropriate ω, the user needs to determine if the domain is such that the cluster may disappear and re-appear over the same spatial extent within a certain period of time. In such cases selecting an $\omega > 1$ will be more appropriate. However, if the domain is such that once the cluster disappears at a certain time stamp or time interval, the same cluster cannot re-appear, then selecting $\omega = 1$ will be sufficient.

6.3.4 Properties of a Spatio-Temporal Polygonal Cluster

A spatio-temporal cluster C_S^T consists of:

1. A set of spatio-temporal (ST)-slices $\left\{c_{\{s_1\}}^{t_1}, c_{\{s_2\}}^{t_2}, \dots, c_{\{s_m\}}^{t_n}\right\}$, where $t_i \in T$ and $\{s_i\} \subset S$ such that $\{s_i\}$ represents the set of polygons $\{p_{s1}^t, p_{s2}^t, \dots, p_{sr}^t\}$ that form each ST-slice at *fixed time interval index* t_i. Henceforth, for simplicity each ST-slice will be represented as $c_{\{s\}}^t$. The space occupied by a ST-slice $\varphi\left(c_{\{s\}}^t\right) = \bigcup_{i=1}^{r} \varphi(p_{s_i}^t)$.

2. A set of temporal-spatial (TS)-slices $\left\{c_{s_1}^{\{t_1\}}, c_{s_2}^{\{t_2\}}, \dots, c_{s_m}^{\{t_n\}}\right\}$, where $\{t_i\} \subset T$ such that $\{t_i\}$ represents the set of polygons $\{p_{s1}^{t1}, p_{s1}^{t2}, \dots, p_{s1}^{tr}\}$ that form each TS-slice *at fixed space index* s_i. Henceforth, for simplicity each TS-slice will be represented as $c_s^{\{t\}}$.

Axiom 1: If $\omega = 1$, then $\varphi\left(c_{\{u\}}^{t1}\right) \cap \varphi\left(c_{\{v\}}^{t2}\right) \neq \emptyset$, i.e. the intersection of the space of two consecutive ST-slices of a spatio-temporal cluster C_S^T cannot be empty. However, if $\omega > 1$ then $\varphi\left(c_{\{u\}}^{t1}\right) \cap \varphi\left(c_{\{v\}}^{t2}\right)$ maybe $\emptyset$, i.e. if $\omega > 1$, then the intersection of the space of two consecutive ST-slices of a C_S^T maybe empty.

Proof: We prove the above axiom in two parts. First, the proof for – If $\omega = 1$, $\varphi\left(c_{\{u\}}^{t1}\right) \cap \varphi\left(c_{\{v\}}^{t2}\right) \neq \emptyset$ is as follows:

The spatio-temporal cluster C_S^T begins by selecting any random spatio-temporal polygon p_S^t, and checking to see if it is a core polygon. If the polygon is indeed a core polygon, its spatio-temporal neighborhood is extracted from the dataset and assigned to the same cluster as the polygon p_S^t itself. Next step is to check if any of the spatial or temporal neighbors are core polygons themselves, and if yes, their neighbors are extracted from the dataset as well, and assigned to the same cluster as p_S^t . This process goes on until no other spatio-temporal polygon gets assigned to the same cluster as p_S^t anymore. Thus, when the first polygon –, polygon p_S^t , gets assigned to the cluster C_S^T, along with its spatio-temporal neighbors, the cluster C_S^T will consist a maximum of three ST-slices. The first ST-slice will consist of only the temporal neighbors of polygon p_S^t at time instant $t-1$, the second ST-slice will consist of the spatial neighbors of the polygon p_S^t and the polygon p_S^t itself (i.e., at time *t*), and the third ST-slice will consist of the temporal neighbors of the polygon p_S^t at time instant $t+1$. As defined before, the temporal neighbors of the polygon p_S^t are the polygons that may exist at time instances $t \pm \omega$, and that occupy *at least* some of the space that was occupied by p_S^t. Thus the intersection of space occupied by any two consecutive ST-slices of a spatio-temporal cluster C_S^T cannot be empty, i.e. $\varphi\left(c_{\{u\}}^{t1}\right) \cap \varphi\left(c_{\{v\}}^{t2}\right) \neq \emptyset,$. Hence proved.

Second the proof for If > 1 , then $\varphi\left(c_{\{u\}}^{t1}\right) \cap \varphi\left(c_{\{v\}}^{t2}\right)$ maybe $\emptyset$ is as follows:

If $\omega > 1$, when the first polygon, polygon p_S^t , gets assigned to the cluster C_S^T, along with its spatio-temporal neighbors, the cluster C_S^T will consist a maximum of $2\omega + 1$ ST-slices. However, as the polygon p_S^t may be designated as a core polygon if it has the required density of Min-Poly polygons taking into consideration its spatial neighbors and temporal neighbors across 2ω time intervals, it may happen that temporal neighborhood of polygon p_S^t may be empty at time interval $t-1$. Thus, in this case, the intersection of ST-slice of cluster C_S^T at time interval t and at time interval $t-1$ will be empty. Hence proved.

Based on axiom 1, the following properties of a spatio-temporal cluster (ST-cluster) can be deciphered:

1. If $\omega = 1$, then a ST-cluster is contiguous in the temporal dimension
2. If $\omega > 1$, then a ST-cluster may lose its temporal contiguity, that is the spatio-temporal cluster may disappear and re-appear at the same location within its lifetime.

6.4 Experimental Analysis

In order to analyze and show the robustness of our algorithm STPC, we first compare its results with other spatio-temporal cluster detection algorithms. Further we study the properties of other parameters of STPC by applying it to the swine flu dataset for the state of California. Finally, in order to show the scalability of our algorithm we have applied it to the crime dataset for the city of Lincoln, NE.

6.4.1 Comparative Analysis using the Drought Dataset

In this section we compare and contrast STPC with 4 other spatio-temporal cluster detection algorithms. These are the MC, CMC, VCoDA, and COT algorithms (see Section 6.2 for a brief description of these algorithms). We have applied all five algorithms to a real-world application that aims at finding moving drought clusters over time and space. For our comparative study, we have used the drought dataset for the state of Nebraska.

Dataset Description: The state of Nebraska has 93 counties. At the end of each week, the U.S. Drought Monitor determines whether each county is in a state of drought based on various measurements of the water cycle. Each county may have regions that experience different levels of drought – severe drought, extreme drought, etc. For our experiments we only take into account whether a county has drought or no drought as a binary decision. 20 weeks of data from Jan 2009 to June 2009 was used for this experiment.

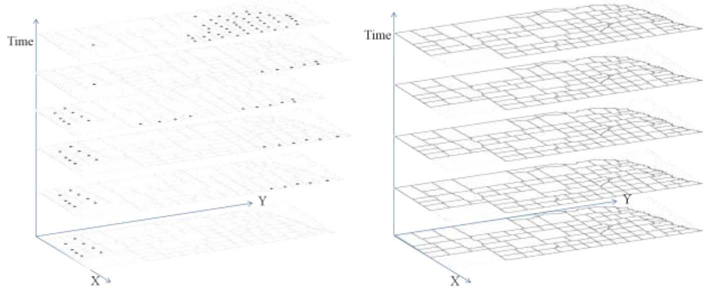


Figure 62: (a) Point representation of drought counties of Nebraska - Dataset for the MC and CMC algorithms (b) Counties of the state of Nebraska – Dataset for the STPC algorithm. The discrete time scale for both the datasets is weekly.

The STPC algorithm has been designed to handle polygonal datasets and thus its input is the set of polygons as shown in Figure 62(b). The MC and the CMC algorithms on the other hand, can only handle point datasets. Furthermore, Both MC and CMC can only handle one class or label of data points at a time. That is, they cannot distinguish between drought and no- drought clusters. Thus, the input needs to be further filtered to contain only the points representing the counties with droughts at each timestamp. Therefore, the input to both these algorithms is shown in Figure 62(a), where each polygon is represented as a point using the centroid of the polygon.

Results: To evaluate our results, we have used as ground truth the drought monitor maps (http://drought.unl.edu/dm/archive.html) produced by the U.S. Drought Monitor (Figure 63). The drought maps for Nebraska from Jan 2009 to June 2009 show that there are three drought clusters and one no-drought cluster.

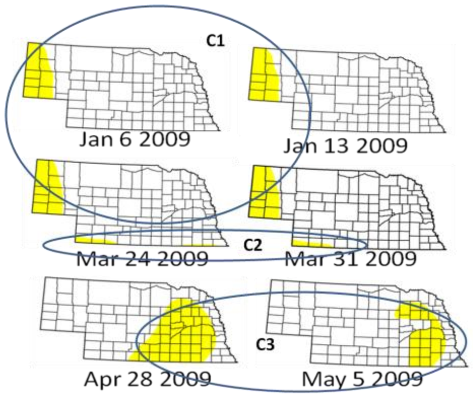


Figure 63: Sample drought monitor maps from http://drought.unl.edu/dm/archive.html showing the three drought clusters.

As defined in Section 6.2.2, the MC algorithm takes as input the parameters ε, θ, and $MinPts$. For our experiments, we have used $\varepsilon = 52\ miles, \theta = 0.5, and\ MinPts = 3$. The CMC algorithm takes as input the parameters ε, k, m, and $MinPts$. For our experiments, we have used $\varepsilon = 52\ miles, m = 3, k = 3, and\ MinPts = 3$. VCoDA takes as input the parameters ε, k, and m. For our experiments, we have used $\varepsilon = 1.0, k = 3, and\ m = 3$. The STPC algorithm is applied using $\varepsilon = 52\ miles, \omega = 1, \alpha = 0, and\ MinPoly = 3$. The COT algorithm only takes as input the parameters ε, and $MinPts$. For our experiments, we have used $\varepsilon = 52\ miles, and\ m = 3$. Please note that $\varepsilon = 52\ miles$ was selected by computing the pair-wise Hausdorff distance between all the polygons in the dataset, and then finding the mode of the all the distance values.

Using the parameters aforementioned, the MC algorithm discovers 5 drought clusters, the CMC algorithm discovers 4 drought clusters, VCoDA discovers 7 drought clusters, STPC discovers 3 drought clusters along with one no-drought cluster, and the COT algorithm discovers 8 drought clusters. Based on the number of drought and no-drought clusters, only the STPC algorithm produces the results same as the ground truth. Furthermore, other than STPC, none other algorithm is able to discover the no-drought cluster because they do not have the ability to distinguish between the objects being clustered based on their non-spatial attributes. Finally, when we compared the clusters discovered by STPC with the ground truth, we found that they were the same clusters. The result obtained by STPC is shown in Figure 64.

Upon comparing the results of STPC with the clusters obtained by other algorithms mentioned above, we found that other algorithms discovered clusters that we indeed part of the clusters discovered by STPC. But none of the other algorithm successfully discovered complete clusters as found at STPC and shown in Figure 64. For example, the trailing end of cluster 1 (C1) shown in Figure 9 was not discovered by any other algorithm as with only one polygon at each time stamp the density condition is not satisfied. However, as STPC is based on the spatio-temporal neighborhood of a polygon rather than only the spatial neighborhood of the polygon, even with a single polygon at each time stamp, the density condition is satisfied. Similarly, the third cluster discovered (C3 in Figure 64) by STPC is divided into two or more clusters by every other algorithm because of the extreme density changes within the cluster from one time stamp to another. The comparison of the results produced by MC, CMC, VCoDA, STPC and COT is further demonstrated in Figure 65 where the charts map the movements of the clusters across space and time. The charts show the number of polygons that belong to a cluster at a particular time stamp. These help to further visualize the density changes occurring within each cluster with the passage of time. VCoDA and COT algorithms discover clusters with constant density only, the MC and CMC algorithms are more robust to fluctuating densities, but even these algorithms are not as flexible as STPC which can capture sudden shifts most effectively.

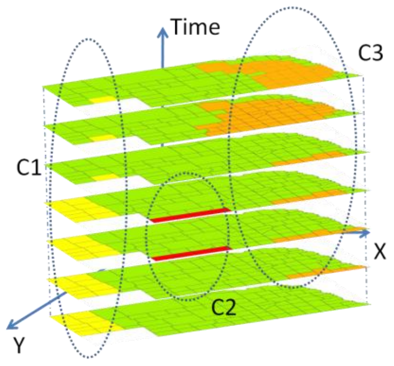


Figure 64: Result of the STPC algorithm – The three smaller clusters are the drought clusters

Furthermore, upon visually inspecting the charts in Figure 65 we can better describe the dynamics of the clusters produced by the various algorithms. For example, for the three clusters tracked by STPC: (1) cluster C1 remains constant for some time and then contracts, (2) cluster C2 remains constant during its lifetime, and (3) cluster C3, after remaining constant for three weeks, contracts to only two polygons and then after expanding a little, suddenly expands across many polygons. This information on the cluster dynamics provides users with another level of insight for decision making. For example based on this dataset one may decide to track cluster C3 more closely to investigate the reasons for the contraction and the subsequent expansion, such as water usage and allocation, and corresponding mitigation policies.

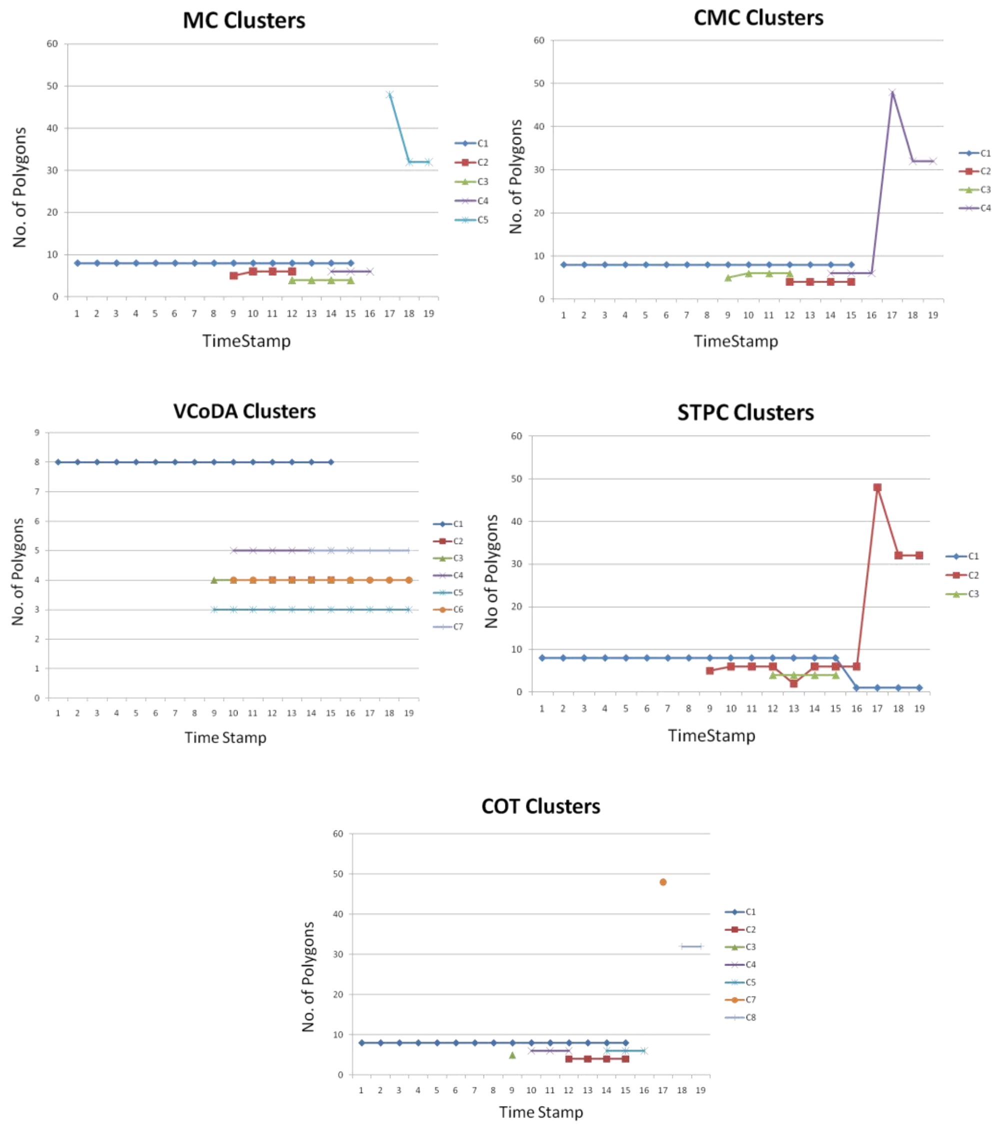


Figure 65: Cluster densities across space and time as discovered by the MC, CMC, VCoDA, STPC, and COT Algorithms for the NE drought dataset

6.4.2 Application on Flu Dataset

In order to show the robustness of our algorithm we have applied STPC to the swine flu dataset for the state of California. In this experiment we observe the properties of the two main parameters of STPC – namely $\varepsilon\ and\ \omega$ where ε dictates the possible extent of the spatial neighborhood and ω defines the possible extent of the temporal neighborhood.

Dataset Description: The dataset for this experiment comprises of the counties of the state of California on a weekly temporal scale from May 28, 2009 to July 16, 2009. Thus the total number of polygons in this dataset is $58 \times 8 = 464$ where 58 is the total number of counties in California, and 8 is the total number of weeks for which the data about the counties is collected. The data that is the non-spatial attributes, for this experiment is the number of new swine flu cases discovered in each county during each time interval. However, we convert the dataset into categorical data by changing the number of new swine flu cases greater than one to 1, and number of new swine flu cases less than one to 0.

Results: For this experiment we applied STPC on the California swine flu dataset using different values of $\varepsilon\ and\ \omega$. The α and $MinPoly$ parameters remain the same for all the experiments as we are using a categorical dataset in which there can be the strong uniformity case where the non-spatial distance between polygons can only be zero, i.e. $\alpha = 0$. We first applied STPC using $\varepsilon = 98\ miles, \omega = 1, \alpha = 0, MinPoly = 3$. Here $\varepsilon = 98\ miles$ is the mode of the pair-wise Hausdorff distance values between the polygons within the dataset. The result is that we discover 5 spatio-temporal clusters of new swine flu cases. The result is shown in Figure 66. Next, we applied STPC using $\varepsilon = 200\ miles, \omega = 1, \alpha = 0, MinPoly = 3$. Here $\varepsilon = 200\ miles$ is the median of the pair-wise Hausdorff distance values between the polygons within the dataset. The result is that we discover one spatio-temporal cluster of new swine flu cases. The result is shown in Figure 67. Finally, we applied STPC using $\varepsilon = 98\ miles, \omega = 5, \alpha = 0, MinPoly = 3$. The result is that we discover 5 spatio-temporal clusters of new swine flu cases. The result is shown in Figure 68. Upon comparing the clusters shown in Figures 66 and 67 we can see that more polygons with new swine flu cases are included in the spatio-temporal cluster shown in Figure 67. Thus when using a smaller ε we can detect more clusters (Figure 66), but there may be some polygons with the same non-spatial attributes as the polygons within the clusters that are not included within the cluster. On the other hand, upon comparing Figures 67 and 68 we can see that the same number of polygons are included in the spatial-temporal clusters in both the cases, even though the number of clusters discovered in Figure 68 are much more than the number of clusters discovered in Figure 67. Finally, upon comparing the clusters shown in Figure 10 and Figure 68, we find the same number of clusters, but the total number of polygons included in the clusters in Figure 68 is more than Figure 66. This can especially be noted in cluster C1 in both the figures. In Figure 68 we can see that cluster C1 is detected much earlier than in Figure 66. This information was lost in the cluster discovered in Figure 66 because of the parameter $\omega = 1$ which stipulates spatio-temporal clusters with temporal contiguity. Thus, if we use a small ε but increase the temporal extent for the spatio-temporal neighborhood of a polygon by selecting $\omega > 1$ there is a greater chance of including more polygons with the same non-spatial attributes within the spatio-temporal clusters discovered.

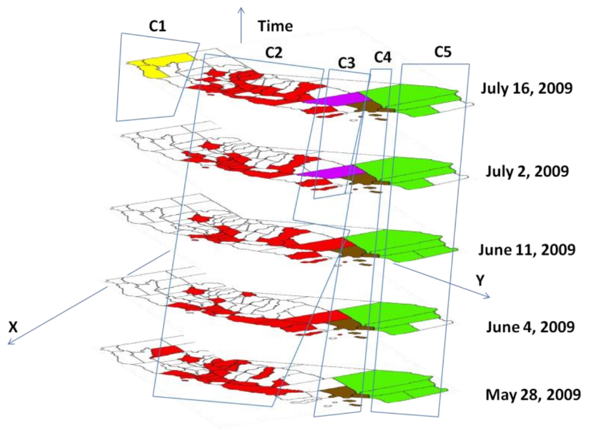


Figure 66: Clusters discovered by STPC with $\boldsymbol{\varepsilon = 98}$ **miles**, $\boldsymbol{\omega = 1}, \boldsymbol{\alpha = 0}$, $\textbf{MinPoly} = \boldsymbol{3}$.

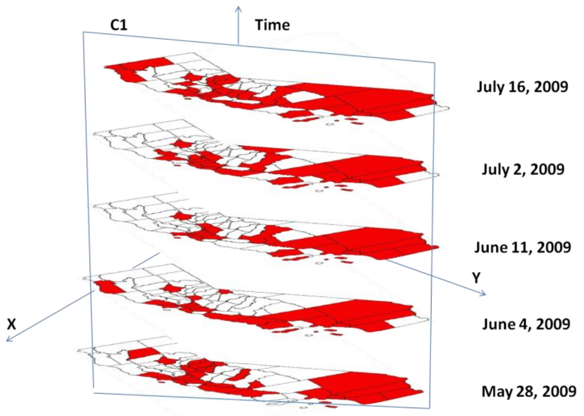


Figure 67: Clusters discovered by STPC with $\boldsymbol{\varepsilon} = \mathbf{200\ miles}$, $\boldsymbol{\omega} = \mathbf{1}$, $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{MinPoly} = \mathbf{3}$.

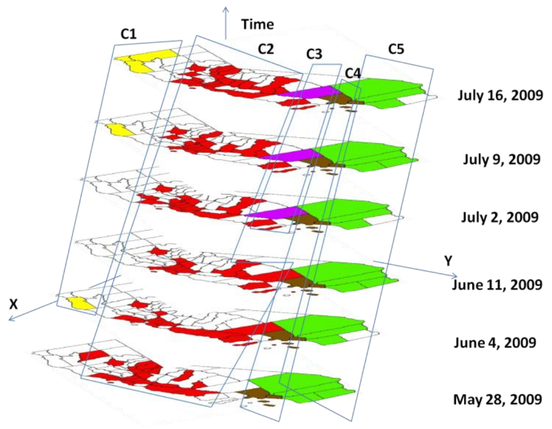


Figure 68: Clusters discovered by STPC with $\boldsymbol{\varepsilon} = \mathbf{98\ miles}$, $\boldsymbol{\omega} = \mathbf{5}$, $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{MinPoly} = \mathbf{3}$.

6.4.3 Application on Crime Dataset

Dataset Description: For this experiment we obtained the dataset from the chief of police of the city of Lincoln, NE, USA. The dataset consists of the time, date, type, and the location of the crime committed over five years between 2005 and 2009. The total number of crimes recorded is 153,404. The city of Lincoln has 186 census block groups. These form the base polygons for our experiments and are shown in Figure 69. The temporal scale used for this set of experiments is daily. The total number of polygons within the dataset is thus 339,450 $(= 186 \times 5 \times 365)$. For each polygon the total number of different types of crimes that occur on each day within the polygon are considered as the non-spatial attributes of the polygons.

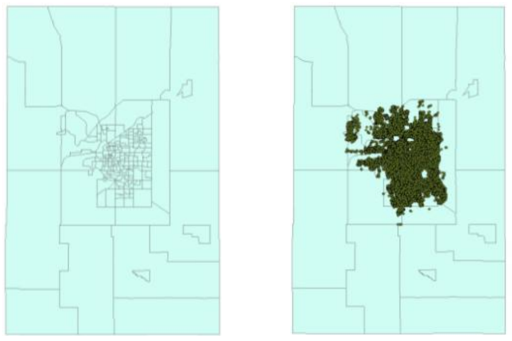


Figure 69: (a) Census block groups in the city of Lincoln, NE (b) Crime locations for the years of 2005 – 2009 in the city of Lincoln, NE.

Results: In the following we show the analysis of the spatio-temporal clusters dsicovered by STPC for one type of crime – assaults. The total number of assault cases in the city of Lincoln from January 2005 until December 2009 is 22,314. The total number of clusters discovered by STPC using different parameter values is listed in Table 18 along with the average number of polygons per cluster and the range of polygons within the clusters. Table 18 shows that the input parameter of $MinPoly$ has a big effect on the clustering results, as the smaller this number is, a a larger number of clusters will be detected. For example, when $MinPoly = 3$, the number of clusters discovered for different ε values is 1132 and 1216. Whereas, when $MinPoly = 20$, the number of clusters discovered are 0 and 121. This is because with a larger $MinPoly$ we are forcing the core polygons to be closer to the center of the entire dataset with a large number of surrounding polygons. On the other hand, with a smaller $MinPoly$, a core polygon may also lie near the periphery of the polygonal dataset. Furthermore, with $MinPoly = 3$, the density may be achieved only by taking into account the temporal neighbors of the polygons, without having any spatial neighbors. The number of clusters discovered is further augmented by the value of ε. A larger ε will allow distant polygons to be included within the same cluster, and therefore allow more number of clusters to be discovered. Thus when MinPoly = 20, and $\varepsilon = 0.65\,miles$, no clusters are detected. This is because no core polygon is discovered that satisfies this criteria.

However, when MinPoly = 20, and $\varepsilon = 1.3\ miles$, 121 clusters are detected, as now there are polygons within the dataset that satisfy this criteria.

Table 18: Assault clusters discovered by STPC using different parameter values

ε	ω	α	MinPoly	# clusters	Average # polygons per cluster	Range of polygons per cluster
0.65 miles	1	5	3	1132	5.5	0.93
0.65 miles	1	5	10	123	15.0	0.78
0.65 miles	1	5	20	0	NA	NA
1.3 miles	1	5	3	1216	8.7	0.96
1.3 miles	1	5	10	301	21.6	0.88
1.3 miles	1	5	20	121	29.8	0.73
1.3 miles	1	5	30	19	35.8	0.41

Further we closely inspect a few selected assault clusters. These are shown in Figure 70. Each cluster is represented by a two-dimensional graph. The x-axis denotes the spatial dimension where each polygon is represented using its identification number. Thus the number 98 along the x- axis represents the space occupied by the polygon with the ID 98. The y-axis shows the temporal dimension of the spatio-temporal clusters. As the crime dataset is from the time period of January 2005 until December 2009 on a daily scale, each day is represented using a unique number. Thus, the number 263 on the y-axis refers to the day of September 20, 2005. To interpret each graph in Figure 70, let us look at Cluster 4. This cluster expands from day 264 to day 269, and covers a total of 26 polygons (IDs: polygons with IDs 49, 56 – 57, 92, 94 – 99, 101 – 105, 107 – 108, 110 – 111, 114 – 117, 119, 122, 134). Furthermore, moving from day to 264 to 265, for example, we can see that only one polygon with ID 95 continues to experience cases of assault, whereas the other polygons (IDs: 56, 94, 97, 119, and 122) do not have any assault cases, instead new polygons (IDs: 99, 104 – 105, and 134) experience assault cases.

From Figure 70 we can see that the clusters 4, 6, and 9 roughly spread across the same set of polygons, however occur a year apart from each other. As they are all assault clusters, we can decipher that these assault cases occurred on the same space during the same time each year for three consecutive years of 2005 to 2007. The polygons involved in these clusters are shown in

Figure 71 which shows the spatio-temporal Cluster 6. A closer inspection of these polygons shows that these polygons are along the heart of the downtown of Lincoln, NE, where most of the bars are located. Furthermore, the time period of mid-September until the beginning of October is when the fall semester at the University of Nebraska-Lincoln which is located close to the downtown area is in full swing, and the weather is most favorable for people to be outdoors. It is interesting to note that the clusters seem to move to an earlier time period (from September to July) in the following two years (2008 and 2009).

6.5 Conclusion and Future Work

We have presented a spatio-temporal clustering algorithm called Moving Polygonal Clustering (STPC) that intrinsically incorporates time in the clustering process of spatio-temporal polygons. The STPC algorithm is based on the density-based clustering principle as this clustering paradigm naturally adapts to concepts such as spatial autocorrelation and Tobler's first law of geography. Furthermore, our algorithm treats time as a first class citizen, and thus gives equal importance to both space and time.

A unique property of our algorithm is that it naturally maintains the history of a cluster. Thus if a cluster fragments into two or more smaller clusters, our algorithm will track the fragmented clusters to the original cluster as long as the fragmented cluster has a temporal neighbor that belongs to the original cluster. Also, if a cluster suddenly contracts and then expands immediately, it will be divided into two or more smaller clusters by MC or CMC. STPC, on the other hand, will be able to capture the sudden movements within a cluster, and will thus be able to retain a unified structure.

As a part of future work, we will test the scalability of our algorithm by taking into consideration the drought dataset for the whole of Unites States of America. Further, we will perform experiments with different values for ω, i.e. change the extent of the temporal neighborhood to see the effect. One possible outcome would be the concatenation of two or more spatio-temporal

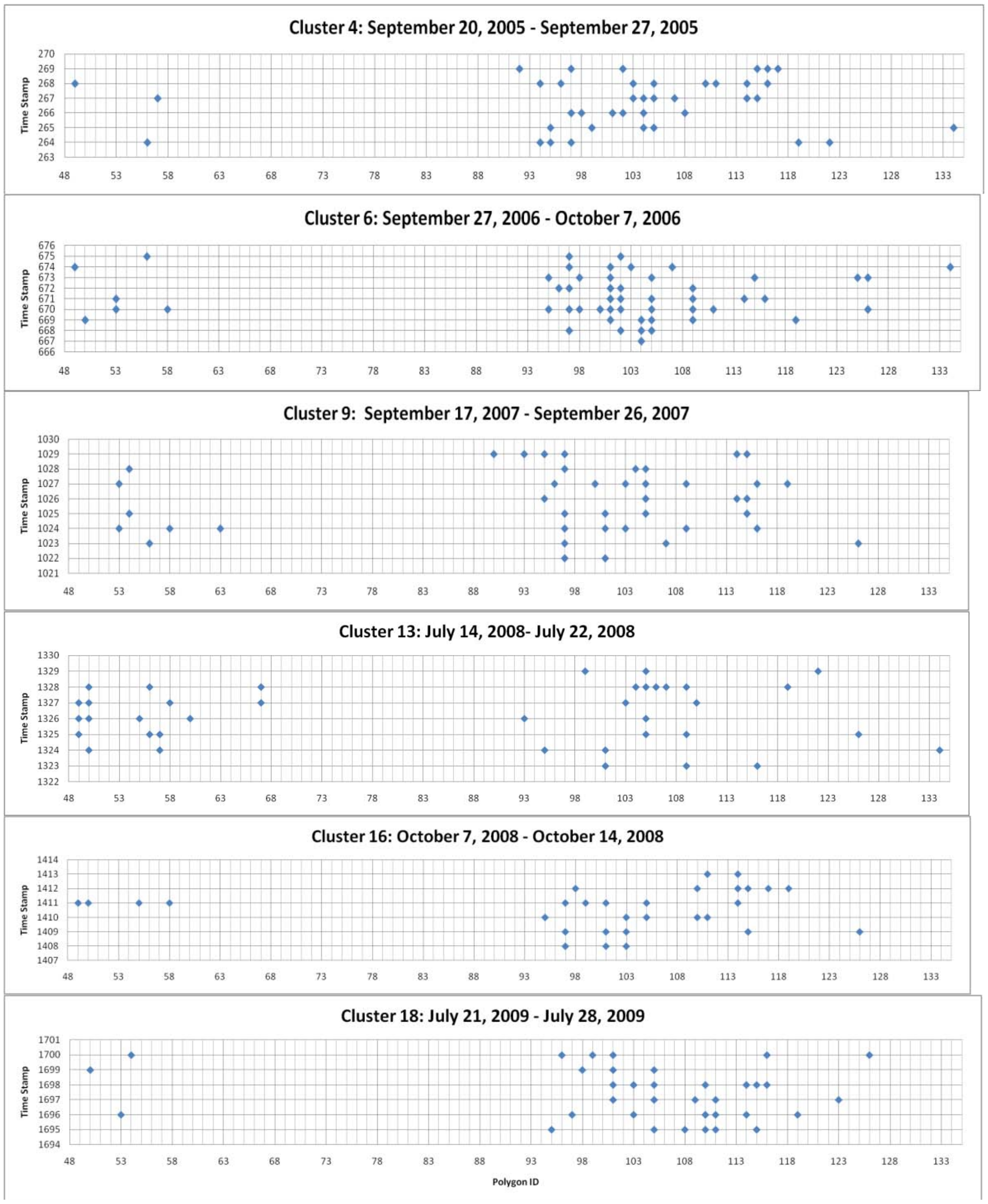
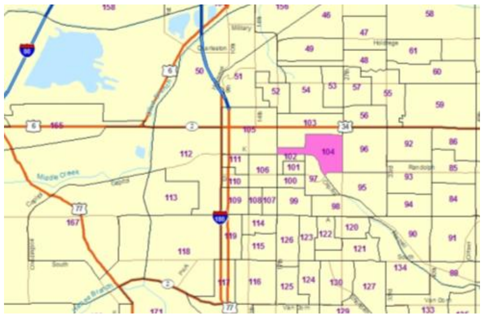
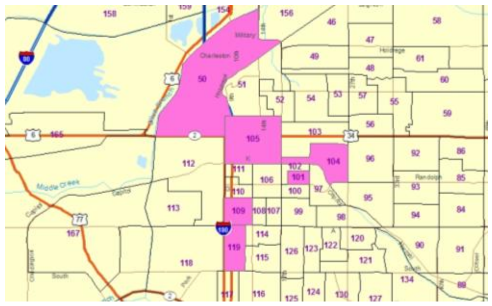
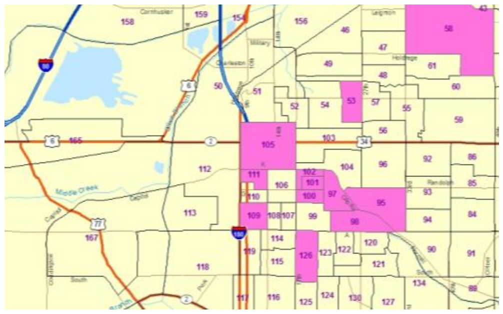
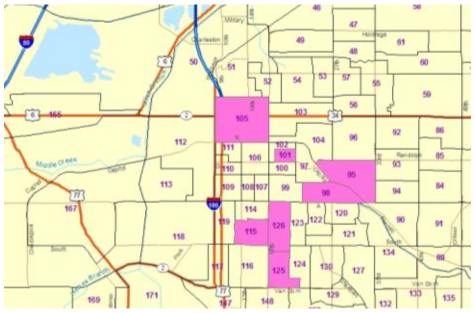


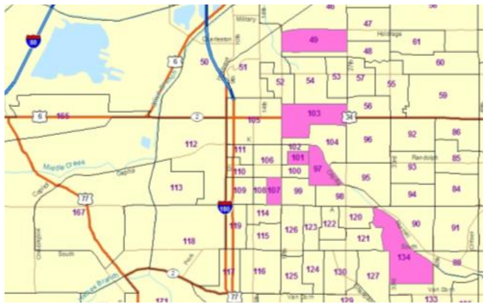
Figure 70: Selected assault spatio-temporal clusters discovered by STPC using the parameter values: $\varepsilon = 1.3\ miles, \omega = 1, \alpha = 5, and\ MinPoly = 30$ with space shown as one-dimension along the x-axis, and time along the y-axis.


September 28, 2006


September 29, 2006


October 2, 2006


October 3, 2006


October 5, 2006

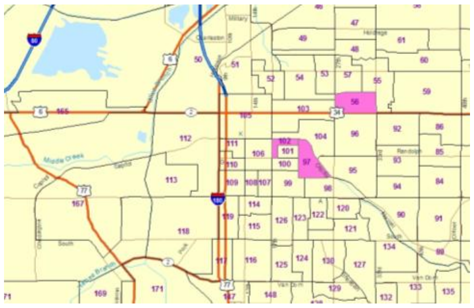

October 6, 2006

Figure 71: The spatio-temporal Cluster 6 in Figure 14 spanning from September 28, 2006 until October 6, 2006

clusters into one spatio-temporal cluster with a time gap in between where the cluster disappears and then re-appears within the same spatial vicinity. Also, currently STPC detects moving clusters across fixed space. We plan to extend STPC to consider moving polygons such as cells of human activities or viruses that could move spatially and change their shapes. In addition, we will extend our framework to detect movements of a cluster in other dimensions than space and time. For example, the varying intensity of drought within spatio-temporal drought clusters.

We are also working on developing a stand-alone java 3D application to visualize the 3-D clusters in the three dimensional space. Next, we will develop a tool to be added in the ArcGIS toolbox that will allow users to form and visualize 3-D clusters within the ArcMap interface.

Publications

This chapter appears in the following:

1. Joshi, D., Samal, A., & Soh, L-. K. (under review), Detecting Spatio-Temporal Polygonal Clusters Treating Space and Time as First Class Citizens, submitted to *GeoInformatica*.

Chapter 7: Analysis of Movement Patterns in Spatio-Temporal Polygonal Clustering

7.1 Introduction

Increasingly spatio-temporal data are being collected as part of systematic environmental monitoring programs. Advances in automated spatial data collection technologies such as geographic positioning systems, satellites, and sensors recording climatic conditions have enabled standardized measurements to be taken for the same location at regular time intervals. A growing interest in monitoring the human and physical environment has led to the production of spatially and temporally referenced data sets (Robertson, Nelson, Boots, & Wulder, 2007). However, the current state-of-the-art in storing and analyzing the spatio-temporal datasets treats time as an additional dimension where the spatial data is stored separately for each time stamp. This approach makes it difficult to further process the data in order to deduce meaningful information from the dataset especially across the temporal dimension. Thus further work, which treats time as a "first-class citizen" needs to be done in order to better organize the spatio-temporal datasets, allowing meaningful information across both the spatial and temporal domains to be derived easily. For this purpose novel algorithms need to be developed to automate the identification, representation and computation of geographic dynamics (Yuan, 2010).

Geographic dynamics refer to the changes that occur across both the spatial and temporal dimensions. For example, within the framework of spatio-temporal data analysis spatial diffusion of various phenomena such as drought, disease, declining/increasing house prices have been extensively studied (Mayer, 2000), (Roehner, 2002). For example when cases of flu are observed within a neighborhood, it is very likely that in the near future more cases of flu will be observed within the surrounding neighborhoods. In order to the study the spread of such events, detecting spatio-temporal clusters of polygons (Joshi, Samal, & Soh, Detecting Spatio-Temporal Polygonal

Clusters Treating Space and Time as First Class Citizens, Under Review) such as counties or census tracts can be an important analysis technique. This is because the study of the clusters can help us in the classification of areas experiencing similar phenomenon across a period of time, along with performing trend analysis and making predictions about the future occurrence of event. However, the current state-of-the-art is lacking in techniques for analyzing the changes that may occur within the spatio-temporal clusters across their spatial and temporal dimensions equally, and formalizing their properties. For example, in order to perform trend analysis and make predictions it will be very helpful if we can study the movements – such as expansion, displacement and convergence – of the spatio-temporal clusters. However, such movements have not been defined formally, neither any other statistics have been defined to quantitatively measure the changes occurring within the spatio-temporal cluster across space and time while treating both space and time as first-class citizens.

In this chapter we present novel techniques for the analysis of polygonal spatio-temporal clusters. For this, we first present a formal framework for treating a polygonal spatio-temporal cluster equally in the spatial and the temporal dimension. Our framework represents and analyzes the polygonal spatio-temporal clusters in two different ways – slicing across the spatial dimension keeping time constant to produce spatio-temporal (ST)-slices), and slicing across the temporal dimension keeping space constant to produce temporal-spatial (TS)-slices). A simple example of a spatio-temporal cluster, and its ST-slices and TS-slice, is presented in Figure 72. This novel outlook allows us to not only observe the changes in the members of the spatio-temporal cluster across the spatial dimension (i.e. observe the movement across the spatial dimension), but also observe the changes in the members of the spatio-temporal cluster across the temporal dimension (i.e. observe the movement across the temporal dimension). For example, by studying the ST-slices of the drought cluster within a region such as the state of Nebraska, we can first analyze the trend of the movement of the cluster, and using this trend make prediction about *where* the drought will move next. Similarly, by studying the TS-slices of the drought cluster within the state of Nebraska, cyclical patterns of the occurrence of drought can be detected for each county individually, and predictions can be made of *when* the county will experience a drought next, therefore enabling the policy makers and the general public to be better prepared.

Next, we formally define the different types of movements a spatio-temporal cluster may undergo during its lifetime based on the ST-slice structure of the cluster, and present measures for detecting the type of movement that has occurred. For this, we have extended the work of Robertson et al (Robertson, Nelson, Boots, & Wulder, 2007) which defines the different types of movements possible for a polygon. The movements for a polygonal spatio-temporal cluster are primarily categorized into four types: 1) displacement, 2) expansion, 3) contraction, 4) no change. Special types of expansion and contraction have also been defined in Section 6.4. While these movements are mutually exclusive, more than one type of movement may occur at the same time, i.e. the cluster may experience expansion and displacement at the same time. Since we focus on the ST-slices in this chapter, we provide only an introduction to the different types of movements that a cluster may undergo during its lifetime based on the TS-slice structure of the cluster as an appendix to this chapter. Further work needs to be done to analyze the TS-slices of the cluster.

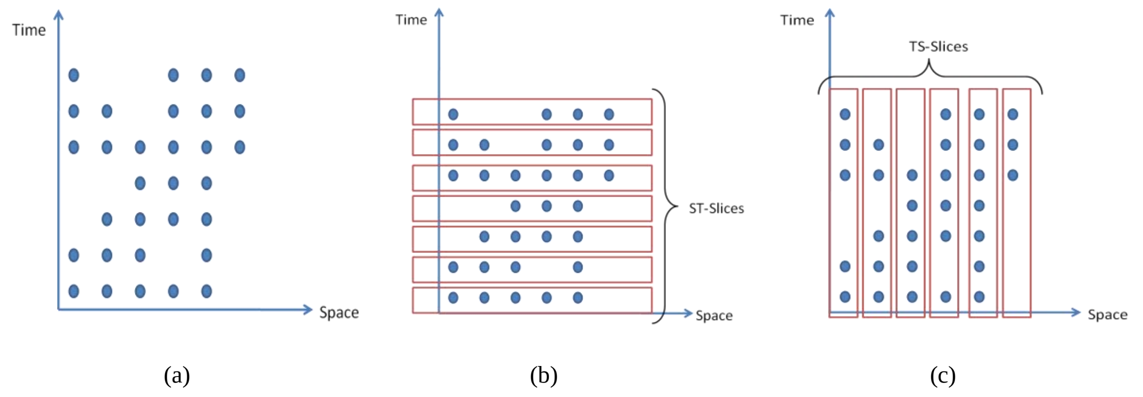


Figure 72: (a) A simplistic spatio-temporal cluster (b) ST-slices of the spatio-temporal cluster (c) TS-slices of the spatio-temporal cluster

Finally, in order to capture the various ST-slice movements of the cluster, and store them efficiently so that further processing may be done on the movement patterns of the cluster, we present the Detecting Movements of Spatio Temporal Clusters (DMSTC) algorithm. DMSTC discovers the different types of movements a spatio-temporal cluster has undergone in its lifetime along with the degree of change the cluster has experienced along with each movement. The output of the DMSTC algorithm is the *movement code* of each ST-slice of the spatio-temporal cluster, along with the statistics that reflect the changes in the 1) area covered by the cluster, the 2) cardinality of the cluster, and 3) the segmentation change in the cluster at each ST-slice or TS-slice. The change statistics can help in identifying the periods of big changes within the lifetime of a spatio-temporal cluster. The movement code, on the other hand, is a vector of vectors where each vector stores the different movements that a ST-slice has undergone. These observations and statistics allow one to summarize, organize, and store data about the spatio-temporal clusters that can further be effectively used for the purposes of trend analysis and predictions. For example, if we have the movement code for the spatio-temporal cluster of cholera in Haiti, and a similar movement pattern is observed in another country, then a prediction on how cholera will spread there can be projected. Thus, the movement code of a spatio-temporal cluster allows us to make comparisons within two or more clusters by factoring out the space and time. This ability makes our framework more robust to handle any application, and make comparisons across both space and time.

In summary, the three main contributions of this chapter are:

1. A formal framework for treating a spatio-temporal cluster equally in the spatial and the temporal dimension;
2. Formal definitions for different types of movements of a spatio-temporal cluster;

3. An algorithm called the Detecting Movements of Spatio Temporal Clusters (DMSTC) algorithm to identify the movements of the spatio-temporal clusters along with the change statistics.

Furthermore, in this chapter, we study the movements of the spatio-temporal clusters discovered by the STPC algorithm (Joshi, Samal, & Soh, Detecting Spatio-Temporal Polygonal Clusters Treating Space and Time as First Class Citizens, Under Review) in three diverse domains – swine flu cluster analysis, drought cluster analysis, and crime cluster analysis. For the swine flu cluster analysis, we study the movements experienced by the swine flu clusters for the state of California in the year of 2009. For the drought cluster analysis application, we study the movements of drought clusters within the state of California for the period of January 2000 until May 2010. Finally, for the crime cluster analysis application, we study the spatio-temporal clusters for the assault crime dataset for the city of Lincoln, NE for a period of five years (2005 – 2009). For all the above applications, we obtain the movement code of the spatio-temporal clusters, along with the change statistics. The movement code of the clusters can be used as input for trend analysis algorithms, as well as prediction algorithms. The change statistics can be used to identify periods of significant change within the spatio-temporal clusters.

The rest of the chapter is organized as follows. Section 7.2 presents a brief background on movements defined for polygons. Section 7.3 defines the various movements possible. Section 7.4 presents the DMSTC algorithm. Section 7.5 discusses our experimental results, and finally Section 7.6 gives our conclusion and future work.

7.2 Related Work

Sadahiro and Umemura (2001) develop a computation model to study the discontinuous changes that may occur in static or fixed polygons over time. They define six types of primitive events that a polygon may experience – 1) generation, 2) disappearance, 3) expansion, 4) shrinkage, 5) union, and 6) division. The change of polygon distributions is decomposed into a combination of these events. For polygon distributions of consecutive time periods a set of events causing the change is deduced. Figure 73 presents the primitive events for polygons that lead to changes in the polygon distributions.

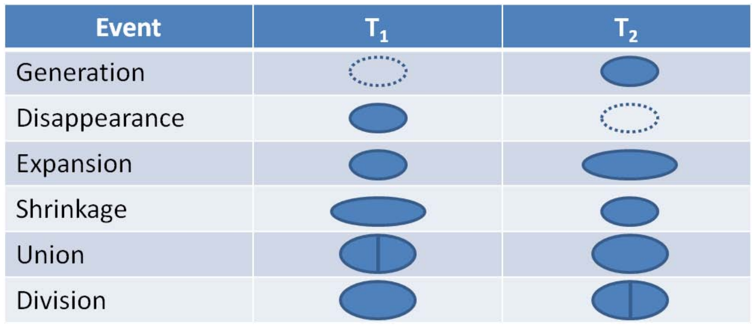


Figure 73: Primitive events for polygons

Robertson et al (2007) extended the work done by (Sadahiro & Umemura, 2001) on analysis of changes in polygons over time, and added movement as a class of polygon change events. They define movement to be a class of events where the polygons are related by proximity, i.e., movement occurs when a polygon does not overlap, but is within a distance threshold of another polygon. In other words, Robertson et al only consider events when a completely new polygon is formed nearby a polygon that existed previously.

Briefly, the following movement patterns are considered from time stamp $t1$ to time stamp $t2$ – Displacement, Convergence, Fragmentation, Concentration, and Divergence (as presented in Figure 74). *Displacement* occurs when a polygon that existed at location l at time instant $t1$ has moved to location $l + d$ at time instant $t2$ where d is the movement distance threshold. *Convergence* occurs when polygons that exist at $t1$ disappear within d of a polygon that expands at time $t2$. Convergence leads to an overall increase in the area covered by the polygons at $t2$. *Fragmentation* is also associated with expansion but occurs when polygons at time instant appear within d of a polygon that has expanded at time $t2$. *Concentration* occurs when polygons at time $t2$ disappear within d of a polygon that has contracted at $t2$. *Divergence* occurs when a polygon appears at time instant $t2$ within distance d of a polygon that has contracted at $t2$.

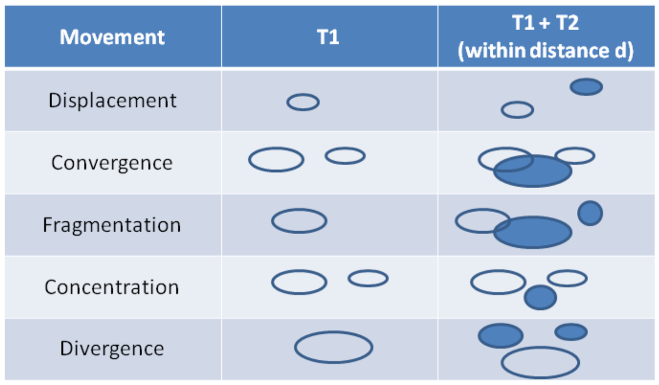


Figure 74: Movement patterns for polygons

7.3 Movements in a Spatio-Temporal Polygonal Cluster

We have adopted the framework presented by (Sadahiro & Umemura, 2001), (Robertson, Nelson, Boots, & Wulder, 2007) as presented before in Section 7.2 to discover the dynamics of a polygonal spatio-temporal cluster that tends to move across space with the passage of time. A spatio-temporal cluster C_S^T consists of:

1. A set of spatio-temporal (ST)-slices $\left\{c_{\{s_1\}}^{t_1}, c_{\{s_2\}}^{t_2}, \dots, c_{\{s_m\}}^{t_n}\right\}$, where $t_i \in T$ and $\{s_i\} \subset S$ such that $\{s_i\}$ represents the set of polygons $\{p_{s1}^t, p_{s2}^t, \dots, p_{sr}^t\}$ that form each ST-slice at *fixed time index* t_i. Henceforth, for simplicity each ST-slice will be represented as $c_{\{s\}}^t$. The space occupied by the ST-slice $c_{\{s\}}^t$ is denoted as $\varphi\left(c_{\{s\}}^t\right) = \bigcup_{i=1}^{r} \varphi(p_{s_i}^t)$.
2. A set of temporal-spatial (TS)-slices $\left\{c_{s_1}^{\{t_1\}}, c_{s_2}^{\{t_2\}}, \dots, c_{s_m}^{\{t_n\}}\right\}$, where $\{t_i\} \subset T$ such that $\{t_i\}$ represents the set of time instances $\{t1, t2, \dots, tn\}$ that form each TS-slice *at fixed space index* s_i. Henceforth, for simplicity each TS-slice will be represented as $c_s^{\{t\}}$. Each TS-slice $c_s^{\{t\}}$ has a beginning time instance denoted as $Begin\{t\}$, and an ending time instance denoted as $End\{t\}$.

We have defined four main types of spatial movements that a polygonal spatio-temporal cluster may undergo when observed in terms of its ST-slices. Figure 75 illustrates how the comparisons between the ST-slices are made in order to discover the spatial movements.

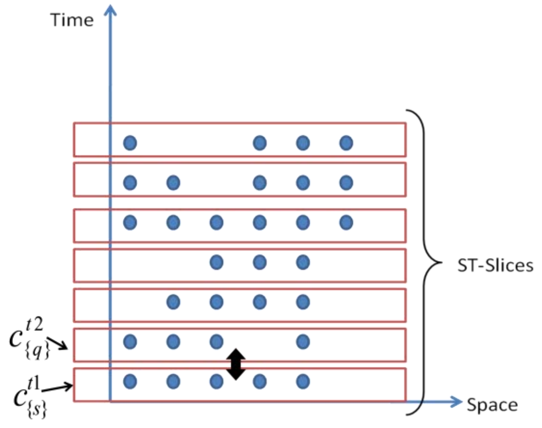


Figure 75: Comparison of ST-slices

M1: Displacement – The polygonal cluster has moved its position at time $t2$ from its position at time 1 , i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$, but $\varphi(c_{\{s\}}^{t1}) \neq \varphi(c_{\{q\}}^{t2})$. Thus **if** $\boldsymbol{\varphi(c_{\{s\}}^{t1}) \neq \varphi(c_{\{q\}}^{t2})}$ **then *movement* = *Displacement***.

M2: Expansion – More polygons have been added to the cluster leading to an overall increase in the total area covered by C_S^T at time $t2$ as compared to time $t1$, i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$, and $area(c_{\{s\}}^{t1}) < area(c_{\{q\}}^{t2})$. Thus **if** $\left(\boldsymbol{area}(c_{\{s\}}^{t1}) < area(c_{\{q\}}^{t2})\right)$ **then *movement* = *Expansion*.**

$area(c_{\{s\}}^{t})$ is the total area of the set of polygons that form the spatio-temporal cluster C_S^T at time slice t. Therefore, $area(c_{\{s\}}^{t}) = \sum_{m=1}^{n} area(p_m^t)$ where $area(p_m^t)$ is the area covered by polygon p_m^t.

Two special cases of expansion may occur. These are classified as two sub-types of movements, and are defined as M2-1 and M2-2.

M2-1: Generation – A new polygonal cluster appears at time $t2$ that did not exist at time $t1$, i.e. $C_S^T = \{\emptyset, c_{\{q\}}^{t2}\}$ where $\emptyset$ represents an empty set. Thus **if** $\left(c_{\{s\}}^{t1} = \emptyset \;\&\; c_{\{q\}}^{t2} \neq \emptyset\right)$ **then *movement* = *Generation***.

M2-2: Merger – Two or more clusters merge together to form a single unified cluster, the cluster is said to merge, i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$ where $c_{\{s\}}^{t1} = \{c_{\{sa\}}^{t1}, c_{\{sb\}}^{t1}\}$ where $c_{\{sa\}}^{t1}$ and $c_{\{sb\}}^{t1}$ are the sub - clusters of $c_{\{s\}}^{t1}$ at time instant $t1$. In other words, the number of spatially contiguous components of the cluster at time $t2$ is less than the number of spatially contiguous components of the cluster at time $t1$. Thus **if** $\left(\boldsymbol{NC}\left(\boldsymbol{c_{\{s\}}^{t1}}\right) > NC\left(\boldsymbol{c_{\{q\}}^{t2}}\right)\right)$ **then *movement* = *Merger*.**

The function $NC\left(c_{\{s\}}^{t}\right)$ computes the number of connected components in ST-slice $c_S^{t'}$ and is described in Algorithm 2 (Figure 76). This algorithm takes as input a graph representation of the ST-slice $c_{\{s\}}^{t}$, and finds the connected components of the graph by performing depth-first search on each connected component [3]. A ST-slice $c_{\{s\}}^{t}$ is represented as a graph by considering each polygon that is a member of ST-slice $c_{\{s\}}^{t}$ as a vertex of the graph, and if two polygons share a portion of their boundaries, then those two vertices are connected by an edge.

M3: Contraction – The polygonal cluster loses some of its constituent polygons at time $t2$ leading to an overall decrease in the total area covered by the cluster, i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$, and $area\left(c_{\{s\}}^{t1}\right) > area\left(c_{\{q\}}^{t2}\right)$. Thus **if** $\left(\boldsymbol{area}\left(\boldsymbol{c_{\{s\}}^{t1}}\right) > area\left(\boldsymbol{c_{\{q\}}^{t2}}\right)\right)$ **then *movement* = *Contraction*.**

Similar to expansion, *two special cases of contraction may occur*. These are classified as two sub-types of movements, and are defined as M3-1 and M3-2.

M3-1: Disappearance – A polygonal cluster that existed at time $t1$ no longer exists at time $t2$, i.e. $C_S^T = \{c_{\{s\}}^{t1}, \emptyset\}$. Thus **if** $\left(\boldsymbol{c_{\{s\}}^{t1}} \neq \emptyset \;\&\; \boldsymbol{c_{\{q\}}^{t2}} = \emptyset\right)$ **then *movement* = *Disappearance*.**

M3-2: Fragmentation – A spatially contiguous cluster at time $t1$ is no longer spatially contiguous at time $t2$, i.e. some of the constituent polygons of the cluster have moved to another cluster at time $t2$, therefore splitting the cluster into two parts, i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$ and $c_{\{q\}}^{t2} = \{c_{\{qa\}}^{t2}, c_{\{qb\}}^{t2}\}$. In other words, the number of spatially contiguous components of the cluster at time $t2$ is greater than the number of spatially contiguous components of the cluster at time $t1$. Thus **if** $\left(\boldsymbol{NC}\left(\boldsymbol{c_{\{s\}}^{t1}}\right) < NC\left(\boldsymbol{c_{\{q\}}^{t2}}\right)\right)$ **, then *movement* = *Fragmentation*.**

M4: No change– The polygonal cluster remains in exactly the same position and consists of the same polygons at time $t2$ as it did in time $t1$, i.e. $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}\}$, but $\varphi\left(c_{\{s\}}^{t1}\right) = \varphi\left(c_{\{q\}}^{t2}\right)$. In other words, when a polygonal spatio-temporal cluster spans across two time instants without undergoing any of the movements listed above from M1 to M3-2, then the polygonal spatio-temporal cluster is said to undergo the *No change* movement. Thus **if** $\boldsymbol{\varphi\left(c_{\{s\}}^{t1}\right) = \varphi\left(c_{\{q\}}^{t2}\right)}$, then ***movement* = *No change.***

Figure 77 presents the movements defined above for a static set of polygons where the polygons themselves do not move in space; however change their attributes with the passage of time. *Please note that a polygonal spatio-temporal cluster can undergo more than one type of movement at any given point of time*. For example, a cluster may undergo displacement and expansion at the same time: additional polygons added to one side of the cluster, causing its centroid to move and its area size to increase.

```
Algorithm 2: $NC$
Input: Graph representation ($G = (V, E)$)of ST-slice $c_j^t$
Output: Number of connected components ($cc$)
Initialize $cc$ = 1
Initialize stack $s$ = empty
Initialize list $visited$ = empty
For each vertex $v \in V$
    If ! ($v \in visited$)
        Call $dfs(G, v, s, visited)$
        Increment $cc$ by 1
    End if
End for
Return $cc$
```

```
$dfs(G, v, s, visited)$
Push $v$ on $s$
While $s$ is not empty
    Pop $v$
    Add $v$ to $visited$
    For each edge $(v, u)$ in $E$
        If $visited$ does not contain $u$
            Push $u$ on $s$
            Add $u$ to $visited$
        End if
    End for
End while
```

Figure 76: The number of connected-components (**NC**) Algorithm

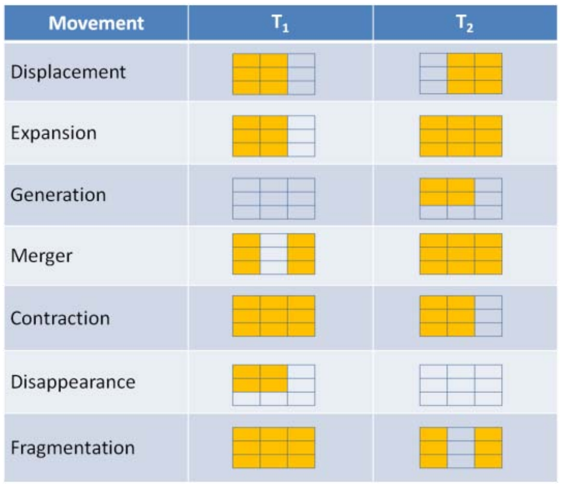


Figure 77: Different types of movements that a polygonal spatio-temporal cluster may undergo

7.4 Detecting Movements Patterns

Based on the definitions of the different types of movements a spatio-temporal cluster can undergo (Section 7.3), we propose an algorithm – Detecting Movements in ST-Clusters (DMSTC) (presented in Figure 78), – that discovers the movement pattern of a spatio-temporal cluster. DMSTC takes as input the spatio-temporal cluster represented as a sequence of ST-slices. At any given time stamp t, the ST-slice $c_{\{s\}}^{t}$ of a cluster C_S^T consists of n polygons $(p_{m=1\ to\ n})$. These are represented as $membership(c_{\{s\}}^{t})$, i.e. the set of polygons that form the spatio-temporal cluster C_S^T at time slice t. The function $|c_{\{s\}}^{t}|$ returns the number of polygons that form the spatio-temporal cluster C_S^T at time slice t. Using this information, DMSTC applies the eight movement tests (M1 to M4) and discovers the movements that the cluster goes through from one ST-slice to another. As the cluster may experience more than one type of movement at the same time, for example – expansion and displacement, the movements experienced by any ST-slice are stored in a vector. Thus, the movement code of the spatio-temporal cluster takes the form of a vector of vectors of movement code for each ST-slice.

Furthermore, DMSTC also measures the changes that occur within the spatio-temporal cluster from one ST-slice to the next. This change is measured using three different types of measures – cardinality change (Δc), area change (Δa), and a segmentation change (Δs). The definitions are included in the algorithm. The *cardinality change* measures the change in the number of polygons that are a member of the cluster at each time stamp. The *area change* measures the change in the total area covered by the cluster from one time-stamp to the next. The *segmentation change* measures the change in the number of connected components within the cluster from one time-stamp to the next. These changes are discovered for all ST-slices of the spatio-temporal cluster, and stored in the parent vector in addition to the movement code vector. Finally, in order to track the movement of the spatio-cluster cluster across space with time, DMSTC also finds the centroid of each ST-slice and stores them in the parent vector.

Using the information generated by DMSTC, the dynamics of the spatio-temporal clusters can be understood in a much better fashion. The movement code can be used to find similar patterns within different spatio-temporal clusters across the world, and may in turn help in predicting the future movements of a cluster. Similarly, tracking the movement of the centroids of a spatio-temporal cluster will enable us to identify the direction of the movement of the cluster. Thus, several cyclical and seasonal patterns can be discovered using this approach.

7.5 Experimental Analysis

In order to evaluate the effectiveness of the DMSTC algorithm we study the movements of the spatio-temporal clusters discovered by the STPC algorithm (Joshi, Samal, & Soh, Detecting Spatio-Temporal Polygonal Clusters Treating Space and Time as First Class Citizens, Under Review)in three diverse domains – swine flu cluster analysis, drought cluster analysis, and crime cluster analysis. For the swine flu cluster analysis, we study the movements experienced by the swine flu clusters for the state of California in the year of 2009. For the drought cluster analysis application, we study the movements of drought clusters within the state of California for the time period of January 2000 until May 2010. Finally for the crime cluster analysis, application we study the spatio-temporal clusters for the assault crime dataset for the city of Lincoln, NE for a time period of five years (2005 – 2009).

```
Algorithm 3 DMSTC
Input: $C_S^T = \{c_{\{s\}}^{t1}, c_{\{q\}}^{t2}, \ldots, c_{\{r\}}^{tm}\}$, The total number of polygons $|P|$ at each time stamp.
Output: Vector of tuples of centroids of connected components, cardinality change ($\Delta$c), area change ($\Delta$a), segmentation change ($\Delta$s), and a vector of movements for each ST-slice $c_{\{s\}}^{t}$.
Initialize Vector $centroids = null$
Initialize Vector $movements = null$
Initialize Vector $\Delta c = null$
Initialize Vector $\Delta a = null$
Initialize Vector $\Delta s = null$
$centroids.add(centroids\ of\ connected\ components\ of\ (c_{\{s\}}^{t}))$
$movements.add(Generation)$
For each ST-slice in $C_S^T$
    $\Delta c.add\left(\frac{\left||c_{\{s\}}^{t+1}|-|c_{\{q\}}^{t}|\right|}{|P|}\right)$
    $\Delta a.add\left(\frac{|area(c_{\{s\}}^{t+1})-area(c_{\{q\}}^{t})|}{area(P)}\right)$
    $\Delta s.add\left(\frac{|NC(c_{\{s\}}^{t+1})-NC(c_{\{q\}}^{t})|}{|P|}\right)$
    $centroids.add(centroids\ of\ connected\ components\ of\ (c_{\{s\}}^{t+1}))$
    $movements.add(GenerateMovementCode(c_{\{s\}}^{t+1}, c_{\{q\}}^{t}))$
End for
Return $centroids, movements, \Delta c, \Delta a, \Delta s$
```

```
$GenerateMovementCode(c_{\{s\}}^{t+1}, c_{\{q\}}^{t})$
Initialize Vector $mc = null$
If $\varphi(c_{\{q\}}^{t}) \neq \varphi(c_{\{s\}}^{t+1})$ then $mc.add(Displacement)$
If $(area(c_{\{q\}}^{t}) < area(c_{\{s\}}^{t+1}))$ then
    $mc.add(Expansion)$
If $(c_{\{q\}}^{t} = \emptyset \;\&\; c_{\{s\}}^{t+1} \neq \emptyset)$ then
    $mc.add(Generation)$
If $(NC(c_{\{q\}}^{t}) > NC(c_{\{s\}}^{t+1}))$ then $mc.add(Merger)$
If $(area(c_{\{q\}}^{t}) > area(c_{\{s\}}^{t+1}))$ then
    $mc.add(Contraction)$
If $(c_{\{q\}}^{t} \neq \emptyset \;\&\; c_{\{s\}}^{t+1} = \emptyset)$ then
    $mc.add(Disappearance)$
If $(NC(c_{\{q\}}^{t}) < NC(c_{\{s\}}^{t+1}))$ then
    $mc.add(Fragmentation)$
If $\varphi(c_{\{q\}}^{t}) = \varphi(c_{\{s\}}^{t+1})$ then $mc.add(No\ change)$
Return $mc$
```

Figure 78: The Detecting Movements in ST-Clusters (DMSTC)Algorithm

7.5.1 Detecting Movement Patterns in Swine Flu Clusters

Dataset Description: The input for this experiment consists of a set of spatio-temporal clusters shown in Figure 79 detected by the STPC clustering algorithm for the swine flu dataset for the counites of the state of California on a weekly temporal scale from May 28, 2009 to July 16, 2009.

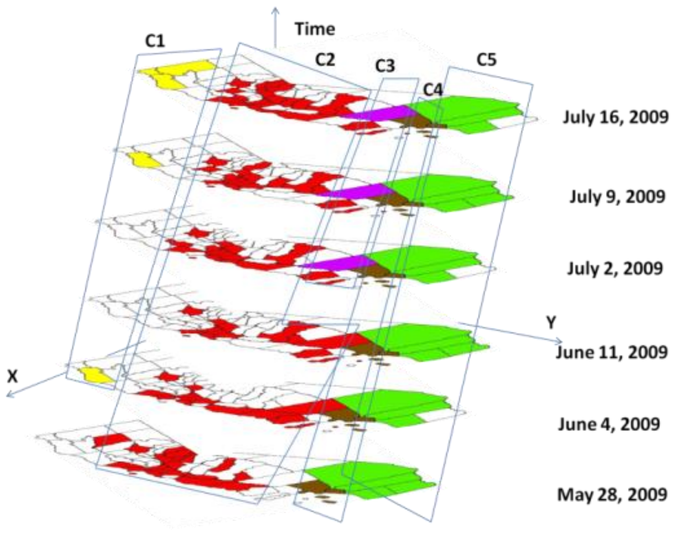


Figure 79: Swine flu clusters for the state of California

Results: The DMSTC algorithm was applied to the 5 spatio-temporal polygonal clusters shown in Figure 79. The movement code for the five clusters as discovered by DMSTC is as follows:

Cluster C1*: <Generation>, <Disappearance>, <Re-Generation>, <Displacement, Expansion>*

Cluster C2*: <Generation>, <Displacement, Contraction, Fragmentation>, <Displacement, Contraction, Fragmentation>, < Displacement, Expansion, Merger >, < Displacement, Contraction, Fragmentation >, < Displacement, Contraction, Fragmentation>, < Displacement, Expansion, Merger >, < Displacement, Expansion>*

Cluster C3*: <Generation>, <No Change>, <Disappearance>*

Cluster C4*: <Generation>, <No Change>, <No Change>, <No Change>, <No Change>, <No Change>, <No Change>, <No Change>*

Cluster C5*: <Generation>, <Displacement, Contraction>, <Displacement, Expansion>, <Displacement, Contraction>, <No Change>, <No Change>, <Displacement, Expansion>, <Displacement, Contraction>*

The movement code discovered for each of the flu clusters shown above can be used in order to make preliminary predictions for the future movement of the cluster. For example, as cluster C3 disappeared after not experiencing any change, cluster C4 is also likely to disappear in the near future as it has not experienced any other change. Cluster C5, on the other hand, is more likely to expand in the near future, and therefore more resources should be assigned to this region in order to prevent the cluster from expanding. The cluster C2 is the most dynamic cluster of all as it sees a lot of movements along with constant fragmentations and mergers happening during its lifetime. While the likelihood of the cluster to contract and fragment is the greatest in the near future, no clear preliminary prediction can be made. In order to predict the future movements of the cluster based on a mathematical model, a trend analysis algorithm for the movement code needs to be developed. This will be a part of our research in the near future.

The three types of change statistics, cardinality change (Δc), area change (Δa), and segmentation change (Δs), for selected swine flu clusters are shown in Figures 80, 81 and 82, respectively. The three different change statistics allow us to identify the periods of significant changes within the lifetime time of a cluster. Looking at Figure 80 we can see that cluster C2 undergoes a lot of change in terms of the number of polygons that are members of the cluster at different time stamps, i.e. it has many more polygons entering and exiting the cluster during its lifetime as compared to Clusters C4 and C5. The peaks at time stamps 4, 7 and 8 indicate that the cluster experiences greatest amount of cardinality change at these time stamps. The cluster C4, on the other hand, does not change its number of polygons once it is "born," and as a result, its Δc is zero between time stamps 2 and 8. While cluster C5 experiences a constant change between time stamps 2 and 5, and then between 7 and 8; that is, the same number of polygons enter and exit the cluster at each of these time stamps.

Figure 81 shows the change experienced by the three clusters in terms of the total area covered by the cluster at each time stamp. Once again we can see that cluster C2 experiences more change than clusters C4 and C5. While cluster C4 does not experience any change in terms of the area covered by the cluster across time stamps 2 and 8, cluster C5 experiences a constant change. It is interesting to note that the cardinality change experienced by cluster C5 is more than the area change experienced by the cluster. This indicates that the ratio of the number of polygons entering and exiting the cluster to the total number of polygons within the cluster is more than the ratio of the change in the area of the cluster to the total area of the cluster.

Figure 82 shows the segmentation change experienced by the three clusters during their lifetime. Once again we can see that cluster C2 fragments or merges a lot in its lifetime, whereas clusters C4 and C5 do not experience any segmentation in their lifetime at all. Therefore, segmentation degree is very different from the cardinality or the area change.

Thus, overall, we can see that cluster C2 is much more dynamic in nature with a lot of changes in its membership, total area covered, and the number of connected components. Therefore, while implementing flu mitigation practices, it will be wise to concentrate on this cluster.

7.5.2 Detecting Movement Patterns in Crime Clusters

Dataset Description: For this experiment we studied a selected set of assault spatio-temporal clusters discovered by STPC from the dataset provided by the chief of police of the city of Lincoln, NE, USA. The dataset consists of the time, date, type, and the location of the crime committed over five years between 2005 and 2009 on a daily scale. The total number of crimes recorded is 153,404. The total number of assault cases in the city of Lincoln from January 2005 until December 2009 is 22,314.

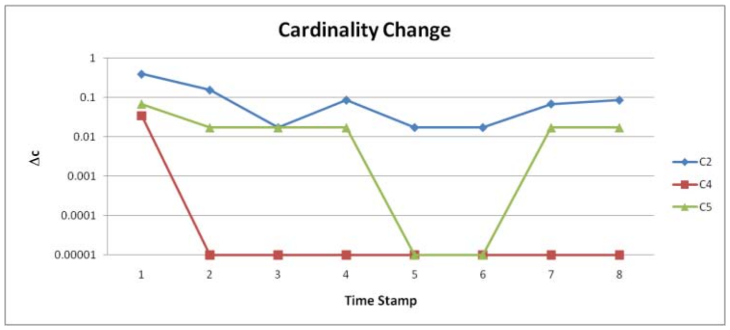


Figure 80: Cardinality change for selected swine flu clusters for the state of California

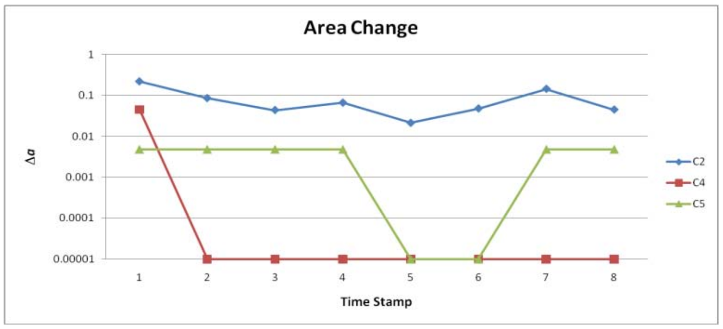


Figure 81: Area change for selected swine flu clusters for the state of California

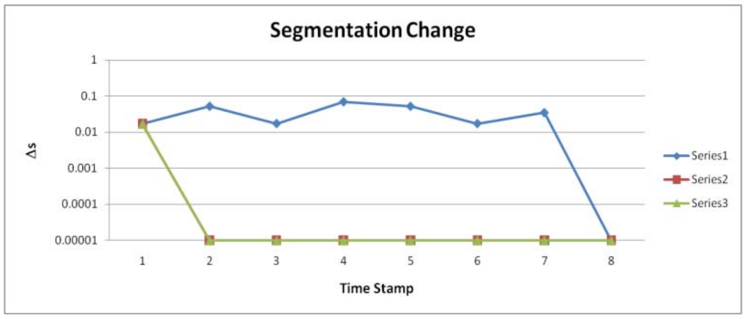


Figure 82: Segmentation change for selected swine flu clusters for the state of California

Results: The results of DMSTC when applied to the selected set of assault clusters are shown in Table 19. Each row in the table shows the cluster ID, the time stamp of the cluster, followed by the change statistics and the movement code of the cluster at that time stamp. The time stamps have a range from 1 to 1857 where 1 refers to January 1, 2005 and 1857 refers to December 31, 2009.

Observing the movement code and the change statistics of the clusters shown in Table 19, we can see that assault clusters do not tend to be distributed in space. This is because there is no fragmentation or merger in the movement code, and also the Δs value is 0 except for the time stamps when the cluster generates and disappears. Furthermore, based on the movement code of these clusters we observe that disappearance of an assault cluster is always preceded by a contraction of the cluster. No conclusion can be derived on the alternation of the Expansion and Contraction movement of the clusters based on this small set of clusters. Further analysis needs to be performed for this purpose along with the implementation of a trend analysis algorithm.

Table 19: Change statistics along with the movement code for selected assault spatio-temporal clusters.

Cluster ID	**Time Stamp**	Δc	Δa	Δs	**Movement**	
0	54	0.0323	0.0011	0.0054	Generation	
0	55	0.0161	0.0005	0.0	Displacement	Contraction
0	56	0.0161	0.0008	0.0	Displacement	Expansion
0	57	0.0005	0.0003	0.0	Displacement	Expansion
0	58	0.0054	0.0006	0.0	Displacement	Contraction
0	59	0.0108	0.0002	0.0	Displacement	Contraction
0	60	0.0269	0.0011	0.0054	Disappearance	
Cluster ID	**Time Stamp**	Δc	Δa	Δs	**Movement**	
6	667	0.0054	0.0002	0.0054	Generation	
6	668	0.0161	0.0006	0.0	Displacement	Expansion
6	669	0.0108	0.0011	0.0	Displacement	Expansion
6	670	0.0323	0.0005	0.0	Displacement	Expansion
6	671	0.0269	0.0013	0.0	Displacement	Contraction
6	672	0.0108	0.0005	0.0	Displacement	Contraction
6	673	0.0108	0.0007	0.0	Displacement	Expansion
6	674	0.0054	0.0003	0.0	Displacement	Contraction
6	675	0.0161	0.0008	0.0	Displacement	Contraction
6	676	0.0161	0.0003	0.0054	Disappearance	
Cluster ID	**Time Stamp**	Δc	Δa	Δs	**Movement**	
14	1333	0.0269	0.0019	0.0054	Generation	
14	1334	0.0054	0.0011	0.0	Displacement	Contraction
14	1335	0.0054	0.0022	0.0	Displacement	Expansion
14	1336	0.0054	0.0007	0.0	Displacement	Contraction
14	1337	0.0054	0.0008	0.0	Displacement	Contraction
14	1338	0.0000	0.0005	0.0	Displacement	Contraction
14	1339	0.0161	0.0004	0.0	Displacement	Contraction
14	1340	0.0108	0.0005	0.0054	Disappearance	
Cluster ID	**Time Stamp**	Δc	Δa	Δs	**Movement**	
15	1365	0.0269	0.0007	0.0054	Generation	
15	1366	0.0108	0.0001	0.0	Displacement	Contraction
15	1367	0.0054	0.0004	0.0	Displacement	Contraction
15	1368	0.0054	0.0005	0.0	Displacement	Expansion
15	1369	0.0161	0.0012	0.0	Displacement	Expansion
15	1370	0.0000	0.0003	0.0	Displacement	Expansion
15	1371	0.0108	0.0007	0.0	Displacement	Contraction
15	1372	0.0323	0.0011	0.0	Displacement	Contraction
15	1373	0.0108	0.0002	0.0054	Disappearance	

7.5.3 Trend Analysis on California Drought Dataset

Dataset Description: For this experiment we studied a selected set of drought spatio-temporal clusters discovered by STPC from the the drought dataset for the state of California for the past 10 years (Jan 2000 – May 2010). The dataset was obtained from drought.unl.edu/dm/dmshps_archive.htm.

Results: Upon the application of the STPC algorithm 15 spatio-temporal polygonal clusters were discovered out of which four clusters were no-drought clusters, and 11 clusters were drought clusters. The DMSTC algorithm was then applied on the 11 spatio-temporal polygonal clusters.

In order to further analyze the movement code of the clusters, we observe the different types of movements that can co-occur. Table 20 lists the different types of movements that can co-occur, and the movements that cannot occur at the same time (these are depicted as NA). The numbers listed in Table 20 are computed based on the movement codes of the 11 drought clusters. We can see that Displacement generally occurs with Expansion or Contraction. While in this case, the number of times Contraction occurs with Displacement is greater than Expansion occurring with Displacement, but the difference is not large enough to differentiate between the two. Further, it is interesting to note that the frequency of co-occurrence of Merger of sub-clusters with Expansion is greater than the frequency of co-occurrence of Merger with Fragmentation. On the other hand, Fragmentation of a big cluster into smaller sub-clusters is generally accompanied with Contraction.

In addition to the movement code, and the change statistics, DMSTC also finds the centroids of the connected components of the ST-slices of the cluster. Using the centroids, we can find the general direction the spatio-temporal cluster is moving towards. The centroids of the ST-slices at each time stamp for the various spatio-temporal drought clusters discovered by STPC are shown in Figure 83. As the cluster moves in time, expanding or contracting and displacing, the centroids show the path of the spatio-temporal clustering the spatial domain with the passage of time. Using this path, trend analysis and future predictions may be made to discover the prospective location of the cluster. For example, it can be noted that the central and southern California experience more drought than northern California. The drought cluster indexed as blue experienced displacements over time and more movements towards May 2008. On the other hand, the droughts in north California tended to be more static and did not experience any movements.

Table 20: Co-occurrence Matrix showing the Eight Movements that occur together for the California drought dataset from Jan 2000 to May 2010.

	G	D	NC	DP	E	C	F	M
Generation (G)	0	NA[a]	NA	NA	NA	NA	NA	NA
Disappearance (D)	NA	0	NA	NA	NA	NA	NA	NA
No change (NC)	NA	NA	0	NA	NA	NA	NA	NA
Displacement (DP)	NA	NA	NA	0	**60**	**77**	**8**	**8**
Expansion (E)	NA	NA	NA	**60**	0	NA	**1**	**5**
Contraction (C)	NA	NA	NA	**77**	NA	0	**7**	**3**
Fragmentation (F)	NA	NA	NA	**8**	**1**	**7**	0	NA
Merger (M)	NA	NA	NA	**8**	**5**	**3**	NA	0

a. NA stands for not applicable, i.e. these two movements cannot occur together.

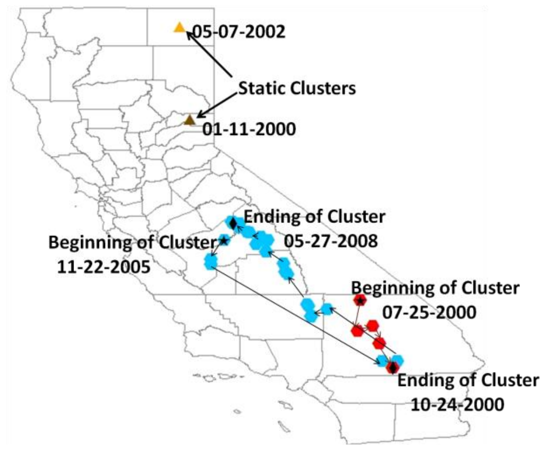


Figure 83:Centroid movement of four different drought clusters across space with time. Two clusters denoted as triangles are static drought clusters, i.e. they do not move across space in time. The red dots and the blue dots respectively show the movement of the other two clusters across space during their respective lifetimes as shown.

7.6 Conclusion and Future Work

In conclusion, we have provided a framework that allows one to view a spatio-temporal cluster as a set of ST-slices or TS-slices. Followed by which we have defined the various movements that a cluster may experience as it moves from one ST-slice to another, and provided tests that will allow the user to easily summarize and store the various movements experienced by a spatio-temporal cluster. Further, we have provided the various change statistics that further help in capturing the dynamics of the cluster in terms of – 1) the change in the number of polygons that are members of the cluster at each time-stamp, 2) the variations in the total area covered by the cluster at each time stamp, and 3) the changes in the number of connected components of the cluster. These statistics along with the movement code of a spatio-temporal cluster are computed using our proposed DMSTC algorithm. In addition, DMSTC also tracks the centroids of the ST-slices of the spatio-temporal clusters capturing the overall direction of the movement of the cluster.

We have applied the DMSTC algorithm to the spatio-temporal clusters detected in three diverse domains – swine flu spread analysis, crime cluster analysis, and drought analysis. With the discovery of the movement code of the clusters belonging to these three distinct domains, we found that while flu clusters are much more dynamic in nature, crime clusters tend to be more limited to a given region without experiencing much distributedness in their lifetime. The drought clusters, on the other hand, tend to move slowly across space and time.

As a part of our future work, we will develop more algorithms that will enable us to capture the dynamics of the spatio-temporal clusters, and analyze them further in order to help the policy makers and the general public be more prepared. For example, more features of the clusters can be discovered with the study of the movement code along with the change statistics. For example with the application of the trend analysis algorithms on the movement code, concrete predictions can be made about the future movements of the clusters. In order to do so, trend analysis algorithms that work with categorical variables need to be developed. This will be a part of our future work, along with the development of other classification and prediction algorithms that will allow us to compare two or more spatio-temporal clusters based on their movement codes and change statistics.

Furthermore, while we have defined the movements of the TS-slices of the spatio-temporal clusters (presented in the Appendix), further work needs to be done in order to analyze the TS-slices. Cyclical and seasonal patterns of a cluster can be discovered by studying the patterns residing within the TS-slices. This also is a part of our immediate future work.

Appendix

We define the four main types of temporal movements that a polygonal spatio-temporal cluster may undergo when observed in terms of its TS-slices. Figure 84 illustrates how the comparisons between the TS-slices are made.

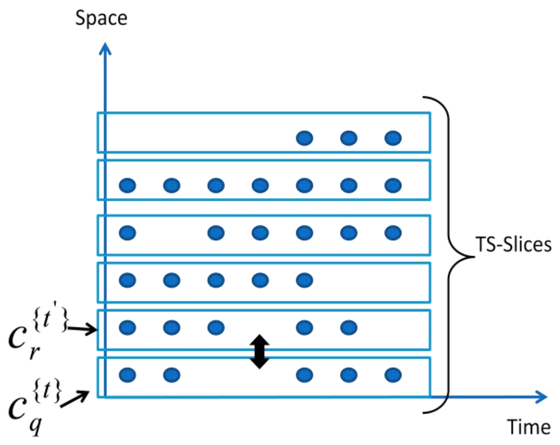


Figure 84: Comparison of TS-slices

M1: Displacement – The beginning or the ending of the polygonal cluster is not the same at location r as compared to location q, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, but $Begin\{b\} \notin \{a\}$ or $End\{b\} \notin \{a\}$. Thus **if** $(\boldsymbol{Begin}\{\boldsymbol{b}\} \notin \{\boldsymbol{a}\} || \boldsymbol{End}\{\boldsymbol{b}\} \notin \{\boldsymbol{a}\})$ **then** ***temporal movement*** **=** ***Displacement***.

M2: Expansion – If the cluster C_S^T spans through greater number of time instances at location r as compared to location q, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, and $|\{a\}| < |\{b\}|$. Thus **if** $(|\{\boldsymbol{a}\}| < \boldsymbol{b}$ **then** ***temporal movement*** **=** ***Expansion*.**

Two special cases of expansion may occur. These are classified as two sub-types of temporal movements, and are defined as M2-1 and M2-2.

M2-1: Generation – A new polygonal cluster appears at location r that did not exist at location q, i.e. $C_S^T = \left\{\emptyset, c_r^{\{b\}}\right\}$ where $\emptyset$ represents an empty set. Thus **if** $\left(c_q^{\{a\}} = \emptyset \,\&\&\, c_r^{\{b\}} \neq \emptyset\right)$ **then *temporal movement* = *Generation***.

M2-2: Merger – The polygonal cluster experiences disappearance and re-generation at location q but it does not exhibit such behavior at location r, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, and $\{a\} = \{t+1, t+2, \dots, t+x-1, t+x, t+x\pm\omega, t+x\pm\omega+1, \dots t+m\} \,\&\&\, \{b\} = \{t+1, t+2, \dots, t+n-1, t+n\}$. Thus **if** $(\{\boldsymbol{a}\} = \{\boldsymbol{t+1}, \boldsymbol{t+2}, \dots, \boldsymbol{t+x-1}, \boldsymbol{t+x}, \boldsymbol{t+x\pm\omega}, \boldsymbol{t+x\pm\omega+1}, \dots \boldsymbol{t+m}\} \,\&\&\, \{\boldsymbol{b}\} = \{\boldsymbol{t+1}, \boldsymbol{t+2}, \dots, \boldsymbol{t+n-1}, \boldsymbol{t+n}\})$ **then *temporal movement* = *Merger***.

M3: Contraction – If the cluster C_S^T spans through lesser number of time instances at location r as compared to location q, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, and $|\{a\}| > |\{b\}|$. Thus **if** $(|\{\boldsymbol{a}\}| > \boldsymbol{b}$ **then *temporal movement* = *Contraction*.**

Similar to expansion, two special cases of contraction may occur. These are classified as two sub-types of movements, and are defined as M3-1 and M3-2.

M3-1: Disappearance – A polygonal cluster that existed at location q no longer exists at location r, i.e. . $C_S^T = \left\{c_q^{\{a\}}, \emptyset\right\}$ where $\emptyset$ represents an empty set. Thus **if** $\left(c_q^{\{a\}} \neq \emptyset \,\&\&\, c_r^{\{b\}} = \emptyset\right)$ **then temporal *movement* = *Disappearance.***

M3-2: Fragmentation – The polygonal cluster experiences disappearance and re-generation at location r but it did not exhibit such behavior at location q, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, and $\{a\} = \{t+1, t+2, \dots, t+n-1, t+n\} \,\&\&\, \{b\} = \{t+1, t+2, \dots, t+x-1, t+x, t+x\pm\omega, t+x\pm\omega+1, \dots t+m\}$. Thus **if** $(\{\boldsymbol{a}\} = \{\boldsymbol{t+1}, \boldsymbol{t+2}, \dots, \boldsymbol{t+n-1}, \boldsymbol{t+n}\} \,\&\&\, \{\boldsymbol{b}\} = \{\boldsymbol{t+}$

$\mathbf{1}, \boldsymbol{t}+\mathbf{2}, \ldots, \boldsymbol{t}+\boldsymbol{x}-\mathbf{1}, \boldsymbol{t}+\boldsymbol{x}, \boldsymbol{t}+\boldsymbol{x} \pm \boldsymbol{\omega}, \boldsymbol{t}+\boldsymbol{x} \pm \boldsymbol{\omega}+\mathbf{1}, \ldots \boldsymbol{t}+\boldsymbol{m}\})$ **then *temporal movement* = *Fragmentation.***

M4: No change– If the cluster C_S^T spans through exactly the same time instances at location r as compared to space q, i.e. $C_S^T = \left\{c_q^{\{a\}}, c_r^{\{b\}}\right\}$, and $\{a\} = \{b\}$. In other words, when a polygonal spatio-temporal cluster spans exactly the same time instances across two consecutive location without undergoing any of the movements listed above from M1 to M3-2, then the polygonal spatio-temporal cluster is said to undergo the *No change* movement. Thus **if** $(\{\boldsymbol{a}\} = \{\boldsymbol{b}\})$, **then *temporal movement* = *No change.***

Publications

This chapter appears in the following:

1. Joshi, D., Samal, A., & Soh, L-. K. (under preparation), Discovering the Movements of Spatio-Temporal Polygonal Clusters, to be submitted to *International Journal of Geographical Information Science*.

Chapter 8: Conclusion

In this research we have addressed the problem of spatial clustering, an important problem in data mining. Specifically, we have focused on clustering geospatial polygons. This is motivated by the fact that most anthropogenic objects in the geospatial space are represented as polygons. The goal is to produce spatially compact and conceptually coherent clusters of polygons taking into account the principles of 1) spatial extent, 2) spatial attributes, 3) spatial relationships, 4) spatial autocorrelation, 5) density-connectivity, 6) spatial constraints, and 7) treating space and time as first-class citizens.

8.1 Summary of Significant Contributions

Specific contributions this research in the area of polygonal spatial clustering are listed below.

- **Dissimilarity function for polygons** – We have developed a dissimilarity function that can efficiently measure the dissimilarity between polygons by integrating both non-spatial attributes and spatial structure and context of the polygons.
- **Density-based polygonal spatial clustering** – We have developed a density-based clustering algorithm for polygons known as P-DBSCAN that extends the density-based concepts for points to polygons taking into account the structural and topological properties of the polygons. We have further extended this algorithm to clustering in the presence of obstacles.
- **Constraint-based polygonal spatial clustering** – We have developed a suite of constraint-based polygonal spatial clustering (CPSC) algorithms that clusters polygons in the presence of user-defined constraints.
- **Spatio-temporal polygonal clustering treating both space and time as first-class citizens** – We have developed a spatio-temporal polygonal clustering algorithm in which

space and time are treated symmetrically. We have also developed an algorithm to identify the different movement patterns within spatio-temporal clusters.

The efficiency and efficacy of our algorithms have been demonstrated using real-life datasets from a variety of application domains including: Environmental Applications (watershed analysis), Public Policy (congressional redistricting, district formation), Climatology (drought analysis), Crime Analysis (Assault cluster analysis), and Spatial Epidemiology (flu analysis).

8.2 Directions for Future Research

This research can be extended in many different directions. Some important challenges are listed below:

1. *Constraint-Based Spatio-Temporal Polygonal Clustering*: The constraint-based spatial clustering algorithm needs to be extended to take into consideration the temporal dimension, along with the physical obstacles and facilitators that may be present.
2. *Analysis of Spatio-Temporal Polygonal Clusters*: More algorithms such as trend analysis algorithms need to be developed to analyze the meaning of the spatio-temporal polygonal clusters discovered.
3. *Visualization of Spatio-Temporal Polygonal Clusters*: Efficient techniques that will allow the results of the spatio-temporal polygonal clustering algorithms to be visualized more intuitively need to be formulated.
4. *Application of Associative Spatio-Temporal Polygonal Clustering in Spatial Epidemiology*: Design algorithms for observing the relationship between two clusters from different datasets but within the same domain. This work would be particularly applied to the field of spatial epidemiology.
5. *Volunteered Geographic Information (VGI) and Citizen Science Applications*: Devise polygonal spatial clustering algorithms for data sets obtained using the VGI systems.

These datasets are fundamentally different as the source of the data will be the users of the system, and issues of confidence and reliability on the data sources will play a key role.

6. *Application of Spatial Polygonal Clustering Algorithms in Biodiversity*. As each region has its own characteristics that play an intrinsic role on the type of life that develops there, space in turn also plays a vital role in the migration of a species. Thus, the application of spatial polygonal clustering on biodiversity datasets will allow us to simultaneously take into account both the spatial features, and the biological features of a region, and discover clusters, which in turn will lead to more accurate results and greater insight.

References

Aamodt, G., Samuelsen, S. O., & Skrondal, A. (2006). A simulation study of three methods for detecting disease clusters. *International Journal of Health Geographics* , 5-15.

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Conference*, (pp. 94-105). Seattle.

Altman, M. (2001). Is Automation the Answer? The Computational Complexity of Automated Redistricting. *Rutgers Computer & Technology Law* , 81-142.

Ankerst, M., Breunig, M., Kriegel, H.-P., & Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. *ACM SIGMOD International Conference on Management of Data*, (pp. 49-60). Philadelphia, PA.

Arkhangel'skii, A., & Pontryagin, L. (1990). General Topology I: Basic Concepts and Constructions Dimension Theory. *Encyclopaedia of Mathematical Sciences* .

Bacao, F., Lobo, V., & Painho, M. (2005). Applying genetic algorithms to zone design. *Soft Comput* , 341–348.

Basu, S., Banerjee, A., & Mooney, R. J. (2002). Semisupervised clustering by seeding. *Proceedings of 19th International Conference on Machine Learning* , (pp. 19-26).

Bodin, L. D. (1973). Democratic representation and apportionment: A Districting Experiment with a Clustering Algorithm. *Annals of New York Academy Sciences* , 209 - 214.

Buchin, K., Buchin, M., & Wenk, C. (2006). Computing the Fréchet distance between simple polygons in polynomial time. *Twenty-Second Annual Symposium on Computational Geometry*, (pp. 80-87). Sedona, Arizona, USA.

Clayton, D. M. (2000). *African Americans and the Politics of Congressional Redistricting.* New York: New York Garland Publishing Co.

Cliff, A. D., Haggett, P., Ord, J. K., Bassett, K. A., & Davies, R. B. (1975). *Elements of Spatial Structure, A quantitative Approach.* Cambridge University Press.

Davidson, I., & Ravi, S. (2005). Clustering with constraints: Feasibility issues and the k-means algorithm. *Proc. of SIAM Int. Conf. of Data Mining.*

Davidson, I., & Ravi, S. (2004). *Towards efficient and improved hierarchical clustering with instance and cluster level constraints.* Department of Computer Science, University at Albany.

Delaunay, B. (1932). Neue Darstellung der geometrischen Krystallographie. *Krystallographie, Vol. 84* , 109-149.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, Vol 1* , 269-271.

Dobkin, D. P., & Kirkpatrick, D. G. (1985). A Linear algorithm for determining the separation of convex polyhedra. *Journal Algorithm, 6,* , 381-392.

Donnelly, K. (1978). Simulations to determine the variance and edge effect of total nearest neighbourhood distance. In Hodder, *Simulation methods in archeology, ed. I.* (pp. 91-95). Cambridge: Cambridge University Press.

Egenhofer, M. J., & Franzosa, R. (1994). On the equivalence of topological relations. *International Journal of Geographical Information Systems* , 133-152.

Egenhofer, M. J., & Mark, D. M. (1995). Modeling conceptual neighborhoods of topological line-region relations. *International Journal of Geographical Information Systems* , 555-565.

Egenhofer, M. J., Clementini, E., & Felice, P. D. (1994). Topological relations between regions with holes. *International Journal of Geographical Information Systems* , 129-144.

Ester, M., Frommelt, A., Kriegel, H.-P., & Sander, J. (2000). Spatial data mining: database primitives, algorithms and efficient DBMS support. *Data Mining and Knowledge Discovery* , 193-216.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Second Int. Conf. on Knowledge Discovery and Data Mining* , (pp. 226-231). Portland, Oregon.

Estivill-Castro, V., & Lee, I. J. (2000a). AUTOCLUST: Automatic Clustering via Boundary Extraction for Massive Point-data Sets. *Proceedings of the 5th International Conference on Geocomputation.*

Estivill-Castro, V., & Lee, I. J. (2000b). AUTOCLUST+: Automatic Clustering of Point Data Sets in the Presence of Obstacles. *Proc. of Intl. Workshop on Temporal, Spatial and Spatio-Temporal Data Mining*, (pp. 133-146). Lyon, France.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning* , 139 – 172.

Furlong, K., & Gleditsch, N. P. (2003). Geographic Opportunity and Neomalthusian Willingness: Boundaries, Shared Rivers, and Conflict. *The Joint Sessions of Workshops European Consortium for Political Research* .

Gardoll, S. J., Groves, D. I., Knox-Robinson, C. M., Yun, G. Y., & Elliott, N. (2000). Developing the tools for geological shape analysis, with regional- to local-scale examples from the Kalgoorlie Terrane of Western Australia. *Australian Journal of Earth Sciences, 47(5)* , 943 - 953.

Grira, N., Crucianu, M., & Boujemaa, N. (2005). Unsupervised and Semi-supervised Clustering: a Brief Survey. *Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence* .

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: an efficient clustering algorithm for large databases. *ACM SIGMOD International Conference on Management of Data*, (pp. 73 – 84). Seattle.

Han, J., & Kamber. (2006). *Data Mining: Concepts and Techniques.* San Fransisco, CA: Morgan Kaufmann Publishers.

Han, J., Kamber, M., & Tung, A. (2001). Spatial clustering methods in data mining: A Survey. In *Geographic Data Mining and Knowledge Discovery* (pp. 1 - 29). Taylor and Francis.

Hayes, B. (1996). Machine Politics. *American Scientist* , 522-526.

Hinneburg, A., & Keim, D. (1998). An efficient approach to clustering in large multimedia databases with noise. *4th International Conference on Knowledge Discovery and Data Mining*, (pp. 58-65). New York.

Huchtemann, D., & Frondel, M. (2010). *Increasing the efficiency of transboundary water manage-ment: a regionalization approach .* Retrieved December 6, 2010, from The Free Library: http://www.thefreelibrary.com/Increasing the efficiency of transboundary water management: a...-a0232178567

Hwang, S.-Y., Chien-Ming Lee, & Lee, C.-H. (2008). Discovering Moving Clusters from Spatio-Temporal Databases. *Eighth International Conference on Intelligent Systems Design and Applications*, (pp. 111-114).

Jeung, H., M.L. Yiu, X. Z., Jensen, C., & Shen, H. (2008). Discovery of convoys in trajectory databases. *Proc. VLDB Endowment, vol. 1, no. 1* , 1068-1080.

Jeung, H., Shen, H. T., & Zhou, X. (2008). Convoy queries in spatio-temporal databases. *ICDE'08*, (pp. 1457-1459).

Jiao, L., & Liu, Y. (2008). Knowledge Discovery by Spatial Clustering based on Self-Organizing Feature Map and a Composite Distance Measure. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B2* .

Joshi, D., Samal, A., & Soh, L. (2009a). A Dissimilarity Function for Clustering Geospatial Polygons. *17th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009).* Seattle.

Joshi, D., Samal, A. K., & Soh, L. (2009b). Density-Based Clustering of Polygons. *IEEE Symposium Series on Computational Intelligence and Data Mining*, (pp. 171-178). Nashville, TN.

Joshi, D., Samal, A., & Soh, L.-.. K. (Under Review). A Dissimilarity Function for Complex Spatial Polygons. *Journal of Geographic Systems* .

Joshi, D., Samal, A., & Soh, L.-.. K. (Under Preparation). Analysis of Movement Patterns in Spatio-Temporal Polygonal Clusters. *GeoInformatica* .

Joshi, D., Samal, A., & Soh, L.-.. K. (Under Review). Detecting Spatio-Temporal Polygonal Clusters Treating Space and Time as First Class Citizens. *GeoInformatica*.

Joshi, D., Samal, A., & Soh, L.-.. K. (Under Preparation). Polygonal Spatial clustering in the Presence of Obstacles . *Transactions in GIS* .

Joshi, D., Soh, L., & Samal, A. K. (2009c). Redistricting Using Heuristic-Based Polygonal Clustering. *IEEE International Conference on Data Mining.* Miami, FL.

Joshi, D., Soh, L.-.. K., & Samal, A. (Under Review). Redistricting using Constrained Polygonal Clustering. *IEEE Transactions on Knowlegde and Data Engineering* .

Kalnis, P., Mamoulis, N., & Bakiras, S. (2005). On Discovering Moving Clusters in Spatio-Temporal Data. *Symposium on Spatial and Temporal Databases* (pp. 364–381). Springer.

Kitzinger, J. (2003). *The Visibility Graph Among Polygonal Obstacles: a Comparison of Algorithms.* University of New Mexico.

Lai, C., & Nguyen, N. T. (2004). Predicting Density-Based Spatial Clusters Over Time. *Proc. of Fourth IEEE International Conf. on Data Mining.*

Macmillan, W. (2001). Redistricting in a GIS environment: Am optimization algorithm using switching points. *Journal of Geographical Systems* , 167-180.

Mann, H., & Fowle, W. B. (1852). *The Common School Journal.* Boston: Morris Cotton.

Mayer, J. D. (2000). Geography, ecology and emerging infectious diseases. *Social Science & Medicine 50* , 937-952.

Neill, D. B., Moore, A., Sabhnani, M. R., & Daniel, K. (2005). Detection of emerging space-timeclusters. *Proc. of 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining.*

Ng, R. T., & Han, J. (2002). CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering* , 1003-1016.

Ng, R., & Han, J. (1994). Efficient and effective clustering methods for spatial data mining. *20th International Conference on Very Large Databases*, (pp. 144 – 155). Santiago, Chile.

Perruchet, C. (1983). Constrained Agglomerative Hierarchical Classification. *Pattern Recognition, 16(2)* , 213 – 217.

Peucker, T. K., & Douglas, D. H. (1975). Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing, 5* , 375-387.

Poone, J. (1997). The Cosmopolitanization of Trade Regions: Global Trends and Implications, 1965–1990. *Economic Geography, Volume 73* , 390-404.

Prager, S. (2010). *UCGIS Working Luncheon Results.* South Carolina: 2010 UCGIS Summer Assembly.

Rao, A., & Srinivas, V. (2005). Regionalization of watersheds by hybrid-cluster analysis. *Journal of Hydrology* , 37-56.

Ravelo, A. C., Andreasen, D. H., Lyle, M., Olivarez Lyle, A., & Wara, M. W. (2004). Regional climate shifts caused by gradual cooling in the Pliocene epoch. *Nature* , 263–267.

Robertson, C., Nelson, T. A., Boots, B., & Wulder, M. A. (2007). STAMP: spatial–temporal analysis of moving polygons. *Journal of Geographical Systems* , 207-227.

Roehner, B. M. (2002). *Patterns of Speculation.* Cambridge, UK: Cambridge University Press.

Rote, G. (1991). Computing the minimum Hausdorff distance between two point sets on a line under translation. *Information Processing Letters* , 123-127.

Ruiz, C., Spiliopoulou, M., & Ruiz, E. (2007). C-DBSCAN: Density-Based Clustering with Constraints. *11th Inl. Conf. on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, (pp. 216-223).

Russell, S. J., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach.* Upper Saddle River, N.J.: Prentice Hall.

Sack, J.-R., & Urrutia, J. (2000). *Handbook of Computational Geometry.* North-Holland, Amsterdam.

Sadahiro, Y., & Umemura, M. (2001). A computational approach for the analysis of changes in polygon. *Journal of Geographical Systems* , 137–154.

Schwartzberg, J. (1996). Reapportionment, gerrymanders, and the notion of compactness. *Minnesota Law Review* , 443 – 452.

Shapiro, L., & Stockman, G. (2001). *Computer Vision.* Prentice Hall.

Sheikholeslami, G., Chatterjee, S., & Zhang, A. (1998). WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. *24th Very Large Databases Conference (VLDB 98).* New York.

Shekhar, S., & Zhang, P. (2004). Spatial Data Mining: Accomplishments and Research Needs (Keynote Speech). *GIScience.*

Shekhar, S., Zhang, P., Huang, Y., & Vatsavai, R. R. (2003). Trends in Spatial Data Mining. In H. Kargupta, & A. Joshi, *Data Mining: Next Generation Challenges and Future Directions.* AAAI/MIT Press.

Stolorz, P. (1995). Fast Spatio-Temporal Data Mining of Large Geophysical Datasets. *Proc. Of the First International Conference on Data Mining*, (pp. 300-305).

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the Number of Clusters in a Data Set via the Gap Statistic. *Journal of the Royal Statistical Society* , 411-423.

Tobler, W. (1979). *Cellular Geography, Philosophy in Geography.* Dordrecht, Reidel: Gale and Olsson, Eds.

Tung, A., Hou, J., & Han, J. (2001). Spatial Clustering in the Presence of Obstacles. *Intl. Conf. On Data Engineering*, (pp. 359-367). Heidelberg, Germany.

Turi, R., & Ray, S. (1998). K-means clustering for colour image segmentation with automatic detection of k. *Proceedings of International Conference on Signal and image Processing*, (pp. 345-349). Las Vegas, Nevada.

Wagstaff, K., Cardie, C., Rogers, S., & Schroedl. (2001). Constrained K-Means Clustering with Background Knowledge. *Proc. of 18th Inl. Conf of Machine Learning*, (pp. 577-584).

Wang, W., Yang, J., & Muntz, R. (1997). STING: A Statistical Information Grid Approach to Spatial Data Mining. *23rd Very Large Databases Conference .* Athens, Greece.

Wang, X., & Hamilton, H. J. (2003). DBRS: A Density-Based Spatial Clustering Method with Random Sampling. *7th PAKDD*, (pp. 563-575). Seoul, Korea.

Wang, X., Rostoker, C., & Hamilton, H. (2004). Density-Based Spatial Clustering in the Presence of Obstacles and Facilitators. *Eighth European Conference on Principles and Practice of Knowledge Discovery in Databases* , (pp. 446-458). Pisa, Italy.

Webster, R., & Burrough, P. (1972). Computer-Based Soil Mapping of small areas from sample data. *Journal of Soil Science, 23(2)* , 210 - 234.

Yoon, H., & Shahabi, C. (2009). Accurate discovery of valid convoys from moving object trajectories. *IEEE International Conference on Data Mining Workshops*, (pp. 636-643). Miami, FL.

Yuan, M. (2010, November 17). GIScience Approaches to Understanding Geographic Dynamics. Lincoln, NE.

Zaïane, O. R., & Lee, C. H. (2002). Clustering Spatial Data When Facing Physical Constraints. *IEEE International Conf. on Data Mining*, (pp. 737-740). Maebashi City, Japan.

Zhang, P., Huang, Y., Shekhar, S., & Kumar, V. (2003). Exploiting Spatial Autocorrelation to Efficiently Process Correlation-Based Similarity Queries. *Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases.*

Zhang, T., Ramakrishnan, & Linvy, M. (1996). BIRCH: an efficient data clustering method for very large databases. *ACM SIGMOD International Conference on Management of Data*, (pp. 103 – 114).

Zhang, X., Wang, J., Wu, F., Fan, Z., & Li, X. (2006). A Novel Spatial Clustering with Obstacles Constraints Based on Genetic Algorithms and K-Medoids. *Sixth International Conference on Intelligent Systems Design and Applications* , (pp. 605-610).