Region-restricted clustering for geographic data mining

Joachim Gudmundsson

Marc van Kreveld

Giri Narasimhan

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2006-031

www.cs.uu.nl

ISSN: 0924-3275

Region-Restricted Clustering for Geographic Data Mining

Joachim Gudmundsson*
National ICT Australia Ltd.,
Sydney, Australia

Marc van Kreveld†
Dept. of Information and Computing Sciences,
Utrecht University, The Netherlands

Giri Narasimhan
Florida International University, Miami, USA

Abstract

Cluster detection for a set P of n points in geographic situations is usually dependent on land cover or another thematic map layer. This occurs for instance if the points of P can only occur in one land cover type. We extend the definition of clusters to *region-restricted clusters*, and give efficient algorithms for exact computation and approximation. The algorithm determines all axis-parallel squares with exactly m out of n points inside, size at most some prespepcified value, and area of a given land cover type at most another prespecified value. The exact algorithm runs in $O(nm \log^2 n + (nm + nn_f) \log^2 n_f)$ time, where n_f is the number of edges that bound the regions with the given land cover type. The approximation algorithm allows the square to be a factor $1 + \varepsilon$ too large, and runs in $O(n \log n + n/\varepsilon^2 + n_f \log^2 n_f + (n \log^2 n_f)/(m\varepsilon^2))$ time. We also show how to compute largest clusters and outliers.

1 Introduction

Spatial data mining is concerned with the detection of interesting patterns from large spatial data sets [16, 21, 25]. For instance, if only one data set is considered, patterns may be related to the concepts of clusters, regularities, or outliers. If more than one data set is considered, patterns of interest may be related to co-locations in space. Objects with many scalar attributes can also be seen as a spatial data set by using the attribute values as coordinates.

In contrast, geographic data mining is a type of spatial data mining where objects or features occupy the geographic space (in the literature, the distinction between spatial and geographical data mining is often not made) [28]. It is a form of geographical analysis: the study into the explanations of geographical phenomena. Geographical analysis includes statistical analysis of geographic data, trend analysis (which includes time), and location planning (which involves combining different data themes) as well.

Clustering has been widely studied in data analysis. Several books and overview articles [18, 19, 20] have appeared on the topic, and any book on data mining discusses clustering. Clustering can be hierarchical or partitional, the number of clusters may be specified beforehand or not, and many different cluster methods exist, each with their properties. In the algorithms field, clustering is still a very active area of research; major conferences have papers that discuss clustering nearly every year.

In this paper we study algorithms for clustering in geographic data mining. The objects to be clustered occupy a geographic space, and that space has other relevant aspects as well. We give three examples.

*Funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

†Supported by the Netherlands Organisation for Scientific Research (NWO) through the BRICKS project GADGET.

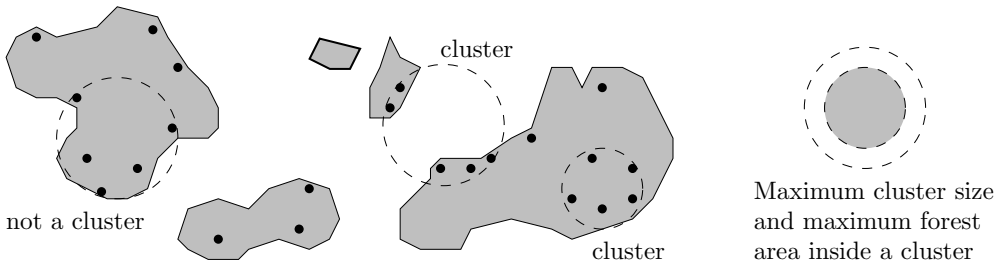

Figure 1: Clustering when five nests are required to form a cluster.

1. Consider a set of points representing bird nests, and imagine a biologist that is interested to what degree the birds of that species seek each other's company when nesting. If the birds always nests in trees, then the only locations where they can have their nests is where the trees are. If the birds are sea birds that nest on islands, then the water in between cannot contain nests. Clustering of bird nest locations should take this into account to decide if a group of nests is a cluster or not. A similar example arises for plants when it is known that the plant can only grow on certain soil types.

2. Consider a set of points representing burglary locations in a city. A cluster of such locations is a group of points that are near to each other. However, if the points occur around the perimeter of a park, then we would like to see this as a cluster as well, because there cannot be buglary locations inside the park. Similarly, car break-ins can only occur where cars may be parked.

3. Consider a set of points representing where lightning has struck. Since lightning is attracted by high buildings or trees, land cover and the height of the land cover should influence the definition of what is considered a cluster.

We abstract the above situations in a simple way. Future research should extend these abstractions to more realistic versions, an issue we discuss in the conclusions section of this paper. Let P be a set of points in the plane, representing the objects that are analyzed for clustering. Assume further that a subdivision into two land cover types A and B is given, and the points of P only occur in the one land cover type, say, B. A cluster is a subset $P' \subseteq P$ with the following properties: (i) P' should be large enough. (ii) The region occupied by P' should not be too large. (iii) The region occupied by P' that is of type B should not be too large. See Figure 1 for an example. We will model (i) by an integer value m, denoting the minimum size of a subset to be called a cluster. We will model the region occupied by P' by a circle or square that contains P'; only smallest circles and squares are of interest. Properties (ii) and (iii) can now be specified by an area value; the value for (ii) should of course be larger than the value for (iii). We give a more formal definition in the next section. Note that the combination of properties (i) and (iii) specifies a lower bound on the density of points from P' in the region where they can be.

According to several sources, clusters are *partitions* of a set of objects. Other sources also use the term cluster for large enough *subsets* of points that are close. For lack of a better term we will also use the term cluster in this paper. In GIS, finding properties of point sets is known as point pattern analysis [27]. Our definition of region-restricted clusters gives a density-based measure for point pattern analysis.

The squares that we find can be used to define larger clusters and of different shapes, by taking the connected components of the union of the squares. Points are then clustered by these connected components. With our definition of a cluster, we can also define outliers. Any point $p \in P$ that does not occur in any subset P' of P with the three properties listed above is an outlier.

Within the research area of (spatial) data mining, one of the most problematic issues is that there are many more potentially interesting patterns than actually interesting patterns. The

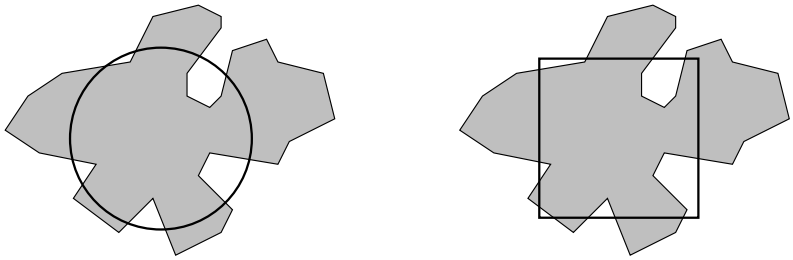

Figure 2: A circle and a square intersecting a simple polygon.

importance of our cluster definition is linked to this. Assume a clustering algorithm does not have property (iii), which is the case for all existing clustering algorithms. If the algorithm must be able to find clusters like burglary locations around a park as well, then the cluster region size must be chosen large enough. However, then many clusters will be found that do not include any park, and after human inspection do not appear to be real clusters. Hence, many non-interesting clusters are generated. Our definition overcomes this problem by separating the extent of the cluster from the density of locations in the relevant areas. Clusters of burglary locations around parks are detected without detecting many false clusters in neighborhoods where there are no parks. The same is true for bird nests in trees.

In Section 2 we formally state the definition of a *region-restricted cluster*, and motivate why we must choose the region to be a square rather than a circle. In Section 3 we give an algorithm that runs in $O(nm \log^2 n + (nm + nn_f) \log^2 n_f)$ time to detect all region-restricted clusters, where n is the number of input points, m is the minimum size of a subset to be a cluster, and n_f is the number of edges that separate the regions of types A and B. As a result of independent interest we develop a data structure of size $O(n_f \log^2 n_f)$ for area containment queries: for any axis-parallel query rectangle, the total area of type B inside it can be reported in $O(\log^2 n_f)$ time. In Section 4 we give an approximation algorithm for the same problem, which runs in time $O(n \log n + n/\varepsilon^2 + n_f \log^2 n_f + (n \log^2 n_f)/(m\varepsilon^2))$. Here, the square may be a factor $(1 + \varepsilon)$ times too large, and may contain a factor $(1 + \varepsilon)$ too much forest area. In Section 5 we give the conclusions and directions for further research.

2 Problem definition

Given a set of disjoint polygonal regions, a distance of interest r, a subset size of interest m, and a set P of points, a cluster is a subset of P of size at least m for which an enclosing circle exists of radius at most $2r$, such that the intersection of this circle and the polygonal regions has total area at most πr^2.

Let us examine the area of intersection of a circle and a single polygon, see Figure 2. It is the sum of square roots, in total linearly many in the number of edges of the polygon that intersect the circle. If we allow the circle to translate slightly, and express its location by its center (x, y) then the area of intersection becomes a function in x and y, whose terms appear in the square roots. To find the position of the circle that minimizes the area of intersection, while intersecting the same set of polygon edges, requires analytical operations that cannot be executed exactly in any reasonable model of computation. In particular, high-degree polynomials must be solved.

If the circle is replaced by an axis-parallel square, the situation is quite different. The function giving the area expressed in the center (x, y) of the square is quadratic, so it can have only a constant number of terms regardless of how many polygon edges intersect the square. It can be evaluated and minimized easily in constant time. To avoid the algebraic issues involved with circles, we define clusters with respect to squares in this paper. It allows us to concentrate on the combinatorial aspects of the problem.

Definition 1 *Given a set of disjoint polygonal regions, a distance of interest s, a subset size of interest m, and a set P of points, a* region-restricted cluster *is a subset of P of size at least m for which an enclosing axis-parallel square exists of side length at most $2s$, such that the intersection of this square and the polygonal regions has total area at most s^2.*

For any region-restricted cluster with more than m points, a subset of m points exists that also is a cluster. Hence, if we determine all subsets of size $\geq m$, a lot of redundant information in the output exists. Therefore, we will give an algorithm for determining all region-restricted clusters of size exactly m. It allows us, for example, to determine all points that participate in *some* region-restricted cluster, and hence, find all points that occur in no cluster. These points can be seen as outliers, if s is such that most points are in some cluster. We also note that the factor 2 that relates the maximum side length of $2s$ to the maximum area of forest of s^2 is not critical. Any other constant than 2 gives the same results; differences in efficiency are only in constant factors.

3 An exact algorithm for region-restricted clusters

Following the analogy of bird nests in trees in forests, we will call the polygonal regions *forests* from now on. The algorithm consists of the following steps. First, for every point $p \in P$, we find the smallest square that has p in the lower left corner and contains exactly m points of P. Second, we trace the collection of all squares that have p on the left side, contain exactly m points, and are smallest. We trace the collection by lowering the square in contact with p on the left side, while adjusting the size such that it is smallest with m points inside. Third, the trace gives us a collection of squares for which we must test whether the side length is at most $2s$ and the area of forest inside is at most s^2. The former test is easy, the latter test is done with a data structure that returns the area of forest inside any query square (or rectangle, for that matter) efficiently. These three steps are described in detail in the next three subsections.

As just mentioned, in Sections 3.1 and 3.2 we show how to find all $O(nm)$ squares that contain exactly m points. This is closely related to the order-m Voronoi diagram of the points in the L_∞-metric. For the L_2-metric, the fastest algorithm takes $O(n \log^3 n + nm \log n)$ expected time and $O(nm)$ space by computing the m-level in an arrangement of planes in 3D after a lifting map [1]. An algorithm of Eppstein and Erickson [13] determines the $O(m)$ nearest points to each point in the L_∞-metric in $O(n \log n + nm)$ time and $O(n \log n)$ space, but requires a RAM model with bit manipulation. It is not clear whether these results can be used to compute the order-m Voronoi diagram in the L_∞-metric. Various other results exist on determining the smallest square or circle that encloses m points [3, 11, 17, 23]; some of these papers consider other measures to minimize on the subset of m points as well. Most of these approaches, however, do not imply the computation of the squares that we require within the same time bound. Furthermore, approaches that compute the order-m Voronoi diagram [1, 3] require $O(nm)$ space and are complicated. We present a simple algorithm that runs in $O(nm \log^2 n)$ time and requires $O(n \log n)$ space.

3.1 Initializing for the sweep

We describe how to determine, for each point $p \in P$, the smallest square that has p in the lower left corner and contains exactly m points of P in $O(nm \log^2 n)$ time. The problem is easy to solve in $O(n^2)$ time: for each point p, find the $(m-1)$-th nearest point in the upper right quadrant of p with respect to the L_∞-metric, using the linear time selection or median finding algorithm of Blum et al. [6, 10]. Our solution is more efficient if m is considerably smaller than n, which is the case in many realistic situations in spatial data mining.

Our solution is based on range query data structures. For each point $p \in P$, we grow a square whose lower left corner is fixed at p, and detect the next point of P that will be inside. After $m-1$ steps, we have the desired square for p. The next point to be inside is determined by four queries, see Figure 3. One query finds the first point reached when the top side of the square is

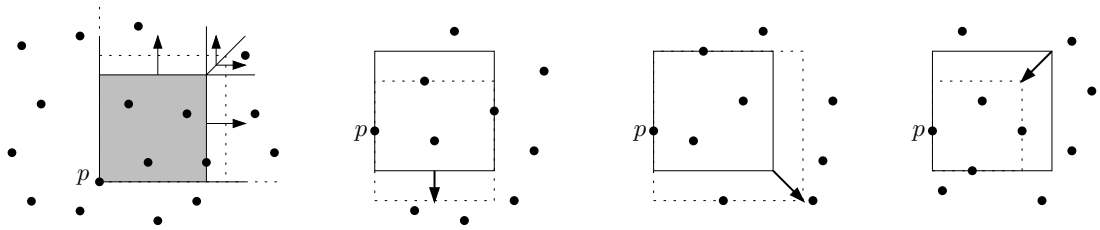

Figure 3: Left, finding the next point inside a growing square by four queries. Right, situations 1, 2, and 3, where the next situations will be 2, 3, and 1, respectively.

translated vertically upwards, a second query finds the first point reached when the right side of the square is translated horizontally to the right, and the third and fourth queries find the lowest and leftmost points in wedges, each bounded by two lines through the upper right corner of the square. The lowest point is found in the wedge bounded by a vertical line and a line with slope 1, and the leftmost point is found in the wedge bounded by a horizontal line and the same line with slope 1. One of the four answers gives the next point that will enter the growing square. With the new point inside, we have the next square, and we perform the same four queries again, but now based on the new, larger square.

The first two queries can easily be solved using a standard, two-dimensional orthogonal range tree [12]. If we apply fractional cascading [9], we get a query time of $O(\log n)$ using a data structure of size and preprocessing time $O(n \log n)$.

The third and fourth queries can be answered using a binary search tree on P sorted by $x - y$, so that a line with slope 1 has the points above and left of it in the left part of the tree, and the points to the right and below it in the right part of the tree. Every internal node of the tree is augmented with a priority search tree [12, 24]. This structure allows us to answer the third and fourth queries in $O(\log^2 n)$ time, using a structure of size $O(n \log n)$ and preprocessing time $O(n \log^2 n)$. Fractional cascading cannot be used to improve the query time for this structure.

For all points $p \in P$, we will perform $O(nm)$ queries in total, and hence the total query time is $O(nm \log^2 n)$. The preprocessing time is $O(n \log^2 n)$ and the storage requirements are $O(n \log n)$. Depending on the machine model used, slight variations on these bounds are possible, for which we refer to [2].

3.2 Sweeping squares along points

We show how to find all "interesting" squares that have a point $p \in P$ on the left side and contain exactly m points inside. To this end, we sweep, grow and shrink the square while keeping its left side in contact with p. The previous section showed how to find the first interesting square for the sweep.

There are three ways in which the sweep can advance: 1. The square translates vertically downward. 2. The square grows to the bottom right. 3. The square shrinks from the top right. We describe these situations in more detail; see Figure 3.

1. The square translates vertically downward when it is in contact with p on the left side and some other point of P on the right side, and continues until either the top side of the square reaches a point of P, or the bottom side of the square reaches a point of P. In the former case we go to situation 2, and in the latter case we go to situation 3.

   All squares during the translation are interesting, in the sense that they may give rise to a region-restricted cluster.

2. The square grows to the bottom right when it is in contact with p on the left side and some point of P on the top side. We cannot lower the top side, or else the square would contain only $m - 1$ points. So we let it grow to the bottom right, until either the right side or the

bottom side reaches a point of P. In the former case we go to situation 1, and in the latter case we go to situation 3.

When the right or bottom side reaches a point of P, the square contains $m+1$ points and therefore is not interesting. As soon as we proceed in situation 1 or 3 we lose the $(m+1)$-th point again. Other squares during the growing are also not interesting, because they properly contain a square with m points inside.

3. The square shrinks from the top right when it has p on the left side and some point of P on the bottom side (but no point of P on the top or right side). The shrinking continues until either the top or the right side reaches a point of P. In the former case we go to situation 2, and in the latter case we go to situation 1.

   Only the final square of the shrinking process is interesting, because it is a subsquare of all others with the same m points inside.

To determine the next event in the sweep efficiently, we use the same two types of data structures as for the initialization of the squares. Translating down requires two queries, namely with the top and bottom sides of the square. Growing to the bottom right requires exactly the same four queries as in the previous section. Shrinking from the top right can be solved with two queries involving standard orthogonal range trees. One query finds the rightmost point in the current square, and the other query finds the topmost point in the current square.

We next analyze how many events occur during the sweep along p. Note that any point that leaves the square through the top side cannot re-enter, and any point that enters the square through the bottom side can do so only once. At the end of situation 2, we always lose a point through the top. At the end of situation 1, we either lose a point through the top or gain a point at the bottom. At the end of situation 3 we lose a point on the bottom side which goes inside the square. Hence, all situations can occur only $O(n)$ times. It follows that for one point $p \in P$, the sweep takes $O(n \log^2 n)$ time, and summed over all points of P this is $O(n^2 \log^2 n)$ time. Preprocessing takes less time, asymptotically. However, we can also show:

Lemma 1 *The running time of all sweeps is $O(nm \log^2 n)$ time.*

Proof. The number of subsets of m points in smallest enclosing squares is $O(nm)$, due to the complexity of order-m Voronoi diagrams in the L_∞-metric [3, 22]. Each event gives a new subset, and each event is handled in $O(\log^2 n)$ time. ∎

We need to do such sweeps for each point $p \in P$ in contact with each side of a square. It is obvious that we can deal with the other sides of squares within the same time bounds.

3.3 A data structure for area intersection queries

The previous section shows how to compute a set of $O(nm)$ subsets of m points that are contained in a smallest square. These squares have a fixed size, but have some x-interval or y-interval of possible locations (for example, the interval of y-coordinates for the top side). The sweeps with the points of P in contact with four possible sides of squares give these intervals. We refer to each such interval as an *interval of squares*.

For all intervals of squares, we first test their size. All that have side length at most s give a region-restricted cluster, even if they are completely covered by forest. All that have side length greater than $2s$ cannot give a region-restricted cluster, because the subset of m points is not close enough. For all intervals of squares whose size is between s and $2s$ we must find out how much forest area is inside each possible location of the square to determine if it forms a region-restricted cluster. In this section we only consider vertical intervals; the horizontal case is symmetric.

We first describe a data structure on the forest regions that, for any query rectangle R, can report the total area of forest inside R. If the forest regions have n_f edges, then the data structure

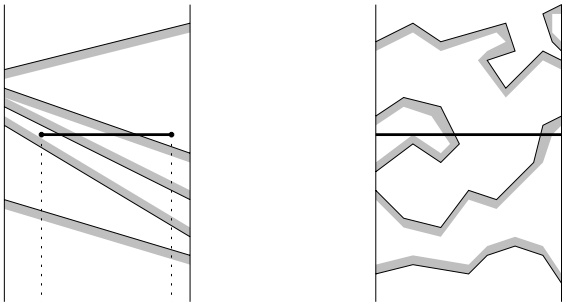

Figure 4: Long forest edges and a short query edge (fat), and short forest edges and a long query edge (fat).

has size $O(n_f \log n_f)$ and answers queries in time $O(\log^2 n_f)$. The structure is based on the hereditary segment tree [8].

The area of a polygon with n edges can easily be computed as the sum of the areas of n quadrilaterals with a horizontal bottom side, vertical left and right sides, and a polygon edge as the top side. Assuming the polygon lies above the x-axis and the vertices are listed clockwise, the area of the polygon $(x_1, y_1), \ldots, (x_n, y_n)$ is:

$$(x_1 - x_n) \cdot (y_1 + y_n)/2 + \sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot (y_i + y_{i+1})/2 \,.$$

Note that trapezoids of edges that bound the polygon locally to the top give a positive area contribution, whereas edges that bound the polygon locally from below give a negative area contribution.

In our situation we also may assume that all forest polygons lie above the x-axis. With every edge of a forest polygon we associate the area of its trapezoid, which can be positive or negative, depending on whether the edge bounds a forest region from above or below. We use the x-intervals of all forest edges to get one-dimensional intervals that are stored on the main tree, which is the same as the main tree of a normal segment tree. The associated structures, stored with all nodes, are used similar to the hereditary segment tree [8].

In a hereditary segment tree, every node ν (internal or leaf) corresponds to some interval I_ν. Let e be some forest edge with forest locally below it. Then e is stored *as a short edge* at every node ν for which an endpoint of e lies in I_ν (in the x-projection). Furthermore, edge e is stored *as a long edge* at every node ν for which I_ν is contained in the x-interval of e, but this does not hold for the parent node of ν (necessarily, e is stored as a short edge at this parent). This is the standard approach for hereditary segment trees.

Each node ν has two associated structures, one for the short edges and one for the long edges stored at ν. Node ν represents a vertical strip $I_\nu \times (-\infty, +\infty)$, see Figure 4. All long edges stored at ν cross the strip from left to right. All short edges come in chains that either connect the left to the right boundary, or have both ends on the left boundary, or have both ends on the right boundary. The forest can always be on either side of long edges and of chains of short edges. Within one strip, the specification of the side that contains the forest may seem inconsistent, e.g., two adjacent long edges may both say that the forest is below it. The specification is only local to each edge, however, and the seeming inconsistency does not give problems with the design of the associated structures.

Assume we query with a rectangle $R = [x_1, x_2] \times [y_1, y_2]$ to determine the area of forest inside R. We will query separately with the horizontal edges of R to determine the aggregated area of forest below those edges. A subtraction then gives the area inside R. So assume that we wish to determine the area of forest in the half-strip vertically below a horizontal edge $[x_1, x_2] \times y_1$. In the tree, we will query the long segments at all nodes on the search paths to x_1 and x_2 (the query

segment is short). Furthermore, we will query the short segments at each node ν for which $[x_1, x_2]$ contains I_ν, but this is not the case for the parent of ν (the query segment is long at ν).

We consider the structure for the edges that are short at ν first. This means that a query will be done with a horizontal line segment that is long at ν. Hence, only its y-coordinate y_1 is of interest for answering the query, which should return the aggregated area of all forest in the strip of ν and below y_1 (and above the x-axis). For every y-coordinate of a vertex in the strip, including the ones generated by intersections of short edges and boundary lines of the strip, the aggerated area has some value. Between two consecutive vertices, the area changes with a function that is quadratic in y. The function itself depends only on the short edges that cross this y-coordinate. Hence, if we imagine a long horizontal line segment starting at the x-axis and moving up, we can determine the aggregated area below the line segment, and maintain the quadratic function when the line segment passes vertices. If there are n_s short segments at ν, then there are $O(n_s)$ events, and we can handle them all in order from bottom to top in $O(n_s \log n_s)$ time. It gives us an associated structure for the short edges of size $O(n_s)$, where each leaf contains a quadratic function that represents the aggregated area below a query y-coordinate. A query is done by finding the appropriate leaf, and evaluating the quadratic function at the query y-coordinate y_1. Obviously, this takes $O(\log n_s)$ time.

Next, we consider the structure for the edges that are long at ν. They have a natural bottom-to-top order because they are disjoint. We use this order to store them in the leaves of a balanced binary search tree. Each internal node stores the edge that is in the rightmost leaf of its left subtree. The tree must be supplied with additional information to be able to answer a query with a short horizontal line segment. For the query in the long segments, we use the clipped version $q = ([x_1, x_2] \cap I_\nu) \times y_1$ of the query segment. Segment q will intersect a consecutive subset of long edges. Either all of these edges intersect q from top-to-bottom (when looking from left-to-right), or they all intersect q from bottom-to-top.

In the preprocessing, consider any leaf, storing a long edge e. The area below any query segment q and edge e, above the x-axis, and between x_1 and x_2, is a quadratic function in x_1, x_2, y_1 that can easily be determined and stored with the leaf. If the forest is locally below e, then the quadratic function will evaluate to a positive value for any query edge, and otherwise to a negative value. Then we compute, bottom-up, the quadratic function for every internal node by taking the sum of the quadratic functions in the two children. At query time, we select all highest nodes that lie strictly between the search paths of the endpoints of q to the leaves, evaluate the quadratic functions stored at each node with the query values x_1, x_2, y_1, and add up the outcomes. If there are n_l long edges at ν, the associated structure has size $O(n_l)$ and query time $O(\log n_l)$.

Using the standard analysis for hereditary segment trees [8], we obtain:

Theorem 1 *A set of disjoint polygons with n_f edges in total can be stored in a data structure of size $O(n_f \log n_f)$, such that for any axis-parallel query rectangle, the area inside can be computed in $O(\log^2 n_f)$ time. The construction time is $O(n_f \log^2 n_f)$.*

We use this structure to test our set of $O(nm)$ vertical intervals of squares. We perform a query with the topmost position, which gives us the area of forest inside. However, instead of returning the *area* of forest inside, we can also obtain the *quadratic function in y* that gives the forest area inside the square if the set of forest edges intersected by the sides of the square is the same. This function will be valid for a small subinterval at the top of the interval that we are testing. When the square is translated down, the combinatorial structure of the forest edges intersecting it will change, and so will the quadratic function in y giving the forest area inside. This is an event in the sweep of the square through the forest regions.

There are two types of event. A corner of the square may pass an edge of a forest region, and a side of the square may pass a vertex of a forest region. We preprocess the forest regions into two data structures that allow us to detect all events on time, before they occur. One is a vertical ray shooting structure in the forest edges. A standard locus approach combined with planar point location solves this; the structure has linear size and logarithmic query time. The other is a segment dragging query, which can be solved using orthogonal range trees with fractional

cascading once again. Alternatively, a result of Chazelle [7] can be used for an optimal solution to segment dragging queries, but this will not improve the overall bounds.

Between any two consecutive events, we consider the quadratic function in y and minimize it, restricted to the relevant subinterval. If the minimum area is at most s^2, we found a region-restricted cluster and report it. Otherwise, we update the quadratic function based on the change of intersected forest edges in $O(1)$ time, and continue the sweep.

For any sweep, there can be $O(n_f)$ events, giving $O(nm \cdot n_f)$ events for all $O(nm)$ sweeps. However, we can show that the number of events during all $O(nm)$ sweeps over the vertical intervals is only $O(n \cdot n_f)$.

Lemma 2 *The $O(nm)$ vertical sweeps of a square have $O(nm + n \cdot n_f)$ events in total.*

Proof. We have $O(nm)$ events for all starts and ends of the sweeps. We consider the two types of intermittent event separately. For the type where a side of the square passes a forest vertex, we consider all vertical intervals caused by a point p on the left side together. From the sweep as described in the previous section, it is clear that any forest vertex is passed only once, because the bottom and top sides of the square (growing, shrinking, and translating) go downward monotonously. Hence, for any point p on the left side, there are $O(n_f)$ events of this type.

For the type of event where a corner of the square passes a forest edge, we observe that the same argument holds for the corners on its left side. These corners go downward monotonously on the vertical line through p. For corners on the right side of the square, for example the top right corner, we cannot use the same argument because there may be $\Omega(n \cdot n_f)$ events for p alone. However, we can use the same argument from the perspective of the point q that lies on the right side: All vertical sweeps with q on the right side are caused by points $p, p', p'', \ldots$, but the intervals they give for the top right corner on the vertical line through q are disjoint. If they were not disjoint, there would be two squares with the same top right corner with m points inside and smallest, which is impossible. Hence, all sweeps that have q on the right side of the square also only give $O(n_f)$ events. The lemma follows. ∎

Theorem 2 *Given a set P of n points in the plane, a set $\mathcal{F}$ of disjoint polygons with n_f edges, a positive integer m, and a positive real s, we can determine all subsets of P of m points for which a smallest enclosing square exists with:*

- *side length at most $2s$, and*
- *total area of polygons from $\mathcal{F}$ inside at most s^2,*

in $O(nm \log^2 n + (nm + nn_f) \log^2 n_f)$ time and $O(n \log n + n_f \log n_f)$ space. To report $O(nm)$ clusters of m points each explicitly, we need additional $O(nm^2)$ time.

Proof. Only the space bound still needs to be proved. We simply note that the $O(nm)$ intervals of squares need not be computed all at once. As soon as we generate a candidate square we test it and report or discard it. So we only need the size of the data structures, which is $O(n \log n + n_f \log n_f)$. ∎

Our solution does not use the property that each polygon in $\mathcal{F}$ is simply-connected, and the result also holds if the polygons have holes.

Remark. For any constant $\delta > 0$, we can also obtain a bound of $O(nm \log^2 n + (nm + nn_f) \log n_f + n_f^{1+\delta})$ time by using a tree of degree $n_f^{1/(2\delta)}$ as the main tree, and using $O(n_f^{1/\delta})$ associated structures for each node (one for each pair of children). This limits the number of associated structures to be queried to $O(1/\delta)$, which is constant. The preprocessing time and size of the data structure increase to $O(n_f^{1+\delta})$. Similarly, we can replace the term $nm \log^2 n$ by $nm \log n + n^{1+\delta}$. Using various data structuring techniques in different machine models, other, slightly different bounds can be obtained as well [2].

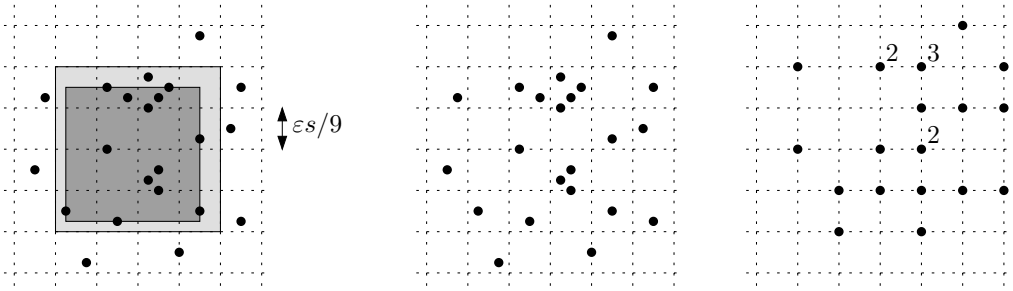

Figure 5: Left, overlaying a grid on P, and enlargement of squares containing exact clusters (dark grey) to squares containing approximate clusters (light grey). Right, snapping to a grid with multiplicity of grid points.

To find outliers, we simply compute the union of the $O(nm)$ squares that give the clusters, and preprocess it for planar point location. Then we query with all points of P. All points that do not lie in the union are outliers. These steps take less time, asymptotically, than the determination of the squares.

Corollary 1 *All region-restricted cluster outliers can be found in* $O(nm \log^2 n + (nm + nn_f) \log^2 n_f)$ *time.*

Remark. For a fixed s and a given integer k, we can determine the largest value of m such that there are at most k outliers. By using values for m of $1, 2, 4, 8, \ldots$, applying the algorithm of Corollary 1 each time until we have more than k outliers, we get an interval $[2^{i-1}, 2^i]$ in which we can do binary search. In total, we need to run the algorithm of Corollary 1 $O(\log k)$ times. Similarly, we can determine the largest cluster for a given value of s.

Remark. For a fixed m and given integer h, we can determine the smallest value of s such that there are at most h outliers. Observe that there are $O(n^2)$ critical values of s for which the clustering may be different. These are the $O(n^2)$ differences of two x-coordinates from points in P and of two y-coordinates from points in P. We can do a binary search on these values without actually computing them, using the selection algorithm for monotone matrices by Frederickson and Johnson [14]. For example, we can determine the median of the $O(n^2)$ x-differences in only $O(n)$ time. The total running time is a factor $O(\log n)$ worse than in Corollary 1.

4 Approximation for the square size

The size of the square, s, is a value that may be user-specified. In any case, the precise value is not crucial, and running the algorithm with a value of s that is, for example, 10% smaller or larger will generally give just as interesting clusters. Therefore, it makes sense to study approximation algorithms, where the size of the square may deviate slightly from what is specified. Approximation allows us to obtain faster running times of the clustering and outlier detection algorithms. At the same time, the number of clusters we find may be significantly smaller than in the exact case. Recall that this is important in spatial data mining.

For a constant $0 < \varepsilon < 1$, the ε-approximate region-restricted cluster reporting problem must determine a set $\mathcal{Q}$ of subsets of P, such that for every region-restricted cluster P' of P, a subset $Q \in \mathcal{Q}$ exists such that $P' \subseteq Q$, the enclosing square S_Q of Q has side length at most $(1+\varepsilon)$ times the side length of the smallest square $S_{P'}$ enclosing P', and area of forest inside S_Q is at most $(1+\varepsilon) \cdot s^2$.

The idea is to overlay a regular grid with spacing $\varepsilon s/9$ over the points of P, snap them to grid points, and only consider squares whose vertices lie on the grid (see Figure 5). If a square S' gives

a region-restricted cluster for a subset P', then our approximation algorithm will find the square S'' whose vertices are snapped outwards onto the grid. The side length of S'' is at most that of S', plus $2\varepsilon s/9$, which is within a factor of $(1+\varepsilon)$ of the side length of S'. The area inside S'' that is not in S' is at most $8s \cdot \varepsilon s/9 + 4(\varepsilon s/9)^2 < \frac{76}{81}\varepsilon s^2$, since $\varepsilon < 1$. We observe that the first, second and third conditions will be satisfied with this idea. We may find approximate region-restricted clusters that do not contain any exact region-restricted cluster, but then they would have been an exact cluster for $s' = (1+\varepsilon) \cdot s$. Snapping to a grid for a factor $(1+\varepsilon)$ approximation has been used in several papers before (see e.g. [5, 17]).

Given P, m, s, and ε, we solve the ε-approximate region-restricted cluster reporting problem as follows. Choose a grid with spacing $\varepsilon s/9$ and place the points of P in the appropriate cells. Then we snap the point to the upper right grid vertex. For each snapped point $p \in P$, we select a subgrid of $(1 + \lceil 18/\varepsilon \rceil) \times (1 + \lceil 18/\varepsilon \rceil)$ grid vertices, where the vertex with p is the upper right corner. The snapped coordinates of p are stored with the selected grid vertices. In total, $O(n/\varepsilon^2)$ grid vertices are selected, many of which may be the same. Selected grid vertices are lower left corners of candidate squares that will be tested.

Note that only grid vertices that are chosen with multiplicity $\Omega(m)$ can be part of a square in which an ε-approximate region restricted cluster lies. There are only $O(n/(m\varepsilon^2))$ such grid points, and hence we need to report no more than $O(n/(m\varepsilon^2))$ clusters to solve the ε-approximate region-restricted cluster reporting problem. For each such grid point, we determine the smallest square that has the lower left corner at this grid point, the upper right corner at another grid point, and contains at least m points. Using the selection algorithm of Blum et al. [6, 10] on the snapped points stored with a grid point, this can be done in $O(n/\varepsilon^2)$ time overall (alternatively, we can use integer sorting [10] by L_∞-distance to the lower left corner).

On the forest edges we build the data structure that was described in the previous section. We query with the $O(n/(m\varepsilon^2))$ squares to determine how much forest is inside. If there is more than allowed, then we discard the square. Otherwise, we report the cluster. We conclude:

Theorem 3 *Given a set P of n points in the plane, a set of disjoint polygons with n_f edges, a positive integer m, a positive real s, and an approximation constant $0 < \varepsilon < 1$, we can solve the ε-approximate region-restricted cluster reporting problem in $O(n \log n + n/\varepsilon^2 + n_f \log^2 n_f + (n \log^2 n_f)/(m\varepsilon^2))$ time.*

Proof. Snapping the points of P takes $O(n \log n)$ time, spreading them over a subgrid takes $O(n/\varepsilon^2)$ time, only $O(n/(m\varepsilon^2))$ grid points are processed further and give rise to a query with a square. Such a query takes $O(\log^2 n_f)$ time after $O(n_f \log^2 n_f)$ preprocessing. ∎

To determine the outliers we can again determine the union of the $O(n/(m\varepsilon^2))$ squares, and locate the points of P. Alternatively, we can store P in a semi-dynamic orthogonal range tree, query with each square, and remove the points of P that lie in any query square.

5 Conclusions and future research

This paper introduced the concept of region-restricted clustering, which is important for geographic data mining. Our clustering definition takes into account the situation where data points may only be possible in certain regions of the plane. It may help to alleviate the problem of detecting many clusters that are not interesting, by using a more appropriate definition of clusters in geographic situations. We have also given efficient algorithms to compute clusters and outliers according to the new definition. A more efficient approximation algorithm was also presented.

A result of independent interest is a new data structure for a set of disjoint polygons with n edges, such that for any query rectangle, the total polygon area inside it can be determined in $O(\log^2 n)$ time. The data structure has size $O(n \log n)$ and can be built in $O(n \log^2 n)$ time.

As we remarked in the introduction, our definition of clusters is restricted in the sense that the shape of a cluster is fixed and its size must be specified. Common clustering methods like k-means, single link, and complete link [20] do not have this restriction. An important topic for

further research is therefore to develop region-restricted versions of these clustering methods. We suggested taking the connected components of the union of squares as a possibility. For clustering methods based on distances, an option would be to use distances according to shortest paths in weighted regions, instead of Euclidean distances [4, 15, 26].

Open problems include developing more efficient algorithms, and extending to the version where the (forest) area requirement of the square is independent of the side length of the square (in other words: where properties (ii) and (iii) of region-restricted clusters specify two values s_1 and s_2 that are unrelated). Other, more general problems of interest include clustering where the points also have attribute values on which clustering is done, and region-restricted clustering in higher-dimensional geographic spaces.

References

[1] P.K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27:654–667, 1998.

[2] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J.E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12:38–56, 1991.

[4] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, 2005.

[5] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[6] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7:448–461, 1973.

[7] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[8] B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[9] B. Chazelle and L.J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(3):133–162, 1986.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[11] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19:474–503, 1995.

[12] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 2nd edition, 2000.

[13] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.

[14] G.N. Frederickson and D.B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM J. Comput.*, 13:14–30, 1984.

[15] L. Gewali, A. Meng, J.S.B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, 1990.

[16] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, San Diego, 2001.

[17] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41:147–157, 2005.

[18] J.A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[19] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[20] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31:264–323, 1999.

[21] K. Koperski, J. Adhikary, and J. Han. Spatial data mining: Progress and challenges. In *Proc. SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1996.

[22] D.T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–487, 1982.

[23] J. Matoušek. On enclosing k points by a circle. *Inform. Process. Lett.*, 53:217–221, 1995.

[24] E.M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[25] H.J. Miller and J. Han, editors. *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, London, 2001.

[26] J.S.B. Mitchell and C.H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38:18–73, 1991.

[27] D. O'Sullivan and D.J. Unwin. *Geographic Information Analysis*. John Wiley & Sons, Hoboken, NJ, 2003.

[28] J. Roddick, K. Hornsby, and M. Spiliopoulou. An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In *TSDM 2000*, number 2007 in Lect. Notes in Art. Int., pages 147–163, Berlin, 2001. Springer.