Visualisation and Analysis of Geographic Information: Algorithms and Data Structures

João Valença

December 12, 2014

CONTENTS

ABSTRACT

1

INTRODUCTION

2

THEORY AND DEFINITIONS

2.1 DEFINITION

In order to be able to quantify representativeness one must first define it. Starting with a set of geographic points N, we must choose a subset, P, that best matches our definition of representativeness. The size of P, however, should not be larger than a given value, k, which will specify how many points we can afford to display.

For any point in N not in P, there must be a point in P that best represents it. In a geographic or geometric plane, with no other deciding factors other than position, this means a distance. This way, the point in P assigned to each point in N will be the one closest to it. This definition of representativeness is referred to as *coverage*.

2.2 DEFINING AS AN OPTIMISATION PROBLEM

The coverage value of a given point, or *centroid*, is given by the radius defined by the distance between itself and the farthest point assigned to it. It can thus be more formally described as:

$$\max_{n \in N} \min_{p \in P} \|p - n\| \tag{1}$$

Where N is the initial set of points, in $\mathbb{R}^2$, and P is the centroid subset. $\|n - p\|$ defines the euclidean distance between point n and the centroid p. The most representative subset, however, will be the one with the minimum value of coverage, since most points covered by the centroids will be closer together, and thus, better represented. We can then finally define our problem as the minimising the coverage:

$$\min_{\substack{P \subseteq N \\ |P| \leq k}} \max_{n \in N} \min_{p \in P} \|p - n\| \tag{2}$$

This is known in the field of optimisation as the *p-centre* problem, and is an example of a facility location problem. It is a *np-hard* problem,

and thus requires some refined approaches in order to be solved in a reasonable time frame.

2.3 DEFINING AS LINEAR PROGRAMMING PROBLEM

A simple and straight-forward approach to the problem is to describe it and implement it using it in terms of integer linear programming. The simplest and most common way to do so is by defining it as such:

$$\text{minimise} \quad D \tag{3}$$

$$\text{subject to} \quad \sum_{j=1}^{N} y_j \leq k \tag{4}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \qquad i = 1, \ldots, N \tag{5}$$

$$\sum_{j=1}^{N} d_{ij} x_{ij} \leq D \qquad i = 1, \ldots, N \tag{6}$$

$$x_{ij} \leq y_j \qquad i = 1, \ldots, N; j = 1, \ldots, N \tag{7}$$

$$x_{ij}, y_j \in \{0, 1\} \qquad i = 1, \ldots, N; j = 1, \ldots, N \tag{8}$$

In this formulation, $y_j = 1$ iff point j is a centroid. $x_i j = 1$ iff the point i is assigned to the centroid j. d_{ij} is the euclidean distance between points i and j. Constraint 4 limits the number of centroids to a number lower than or equal to k. Constraint 5 limits the assignment of one point to more than one centroid. Constraint 6 ensures that all active distances are lower than the limit we are minimising. Constraint 7 stops non-centroid from having other points assigned to them, leaving only the centroids to cover other points.

It is worth noting that this formulation minimises the objective function by selecting the best possible set of centroids, but it does not necessarily output the clearest assignment. In fact, only the farthest point from its centroid has the guarantee that it is connected to its closest centroid.
Every other point, however, can be linked to any centroid so long as it is closer to it than the distance defined by the objective function. It can also produce the result where one centroid is assigned to another centroid, and not itself, as long as they are close enough together. These cases have no effect on the outcome of the final coverage value or the centroid selection, but are rather counter-productive, since we want to minimize the coverage of all centroids and have a cleaner display, with no overlapping of the covered areas.
In order to best display the results, a simple post-process can be applied, where each point will be strictly assigned to the closest centroid. This can be easily computed in polynomial $\mathcal{O}((N-k)k)$ time in case there is a need for a clearer display of the affectation.

2.4 ALGORITHMIC CONCEPTS

2.4.1 *Branch and Bound*

Minimising coverage, as we have seen, is a *np-hard* combinatorial problem. These problems can be approached using Branch and Bound algorithms.

These algorithms solve the problems by recursively dividing the solution set in half, thus *branching* it into a rooted binary tree.
At each step of the subdivision, it then calculates the upper and lower bounds for the best possible value for the space of solutions considered at that node. This is aptly called *bounding*. In the case of a minimisation problem, it would be the upper and lower bounds for minimum possible value for the objective function in the current node. These values are then compared with the best ones already calculated in other branches of the recursive tree, and updated if better.
The idea is to use these values to prune the search tree. This can be done when the algorithm arrives at a node where the lower bound is higher than the best calculated upper bound. At this point, no further search within the branch is required, as there is no solution in the current branch better than one that has already been calculated.
In the case that the global upper and lower bound meet, the algorithm has arrived at the best possible solution, and no further computation must be done.

These algorithms are very common in the field of optimisation and can achieve very fast times, but their performance depends on the complexity of its bounds. Tighter bounds accelerate the process, but are generally slower to compute, so a compromise has to be made in order to obtain the fastest possible algorithm, and fast bound calculation is a priority for most approaches.

2.5 GEOMETRIC CONCEPTS AND STRUCTURES

2.5.1 *Nearest Neighbour*

Any kind of geometry-based algorithms used to solve the problem will quickly face the need to implement a nearest neighbour search. Since we need to find the closest centroid to a given point, this operation will be one of the most used, and so we need a fast and flexible way of determining which of the centroids is closest, in order to reduce overhead.

2.5.2 *Point Location*

A concept commonly associated with nearest neighbour search is point location. Dividing the plane in regions so that it is possible to disregard distant points and direct the search to a smaller number of points makes it so that a lot of nearest neighbour search queries can be done faster, with just a small pre-processing step. Common structures for point location are *k-d trees*as described in [[Cambazard]]. *k-d trees* partition the space using a divide and conquer approach to define orthogonally aligned half-planes in order to speed point location queries to $\mathcal{O}(logn)$ time. These, however need to be periodically updated in order to keep their efficiency and cannot be constructed or de-constructed incrementally without considering this overhead.

2.5.3 *Voronoi Diagrams*

Voronoi diagrams partition the space into regions, which are defined by the set of points in the space that are closest to a subset of those points. Definitions of distance and space may vary, but on our case we will consider the $\mathbb{R}^2$ plane and the euclidean distance.
Such a partitioned plane will look like this:

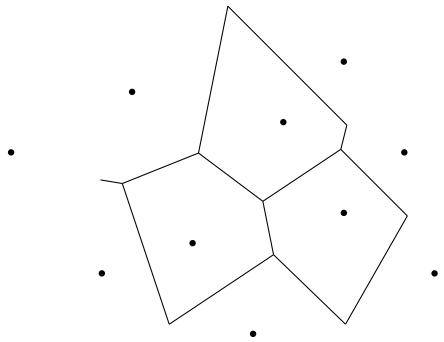

Figure 1: Voronoi Diagram Example

Dashed lines extend to infinity. As we can see, any new point inserted in this plane will be contained inside one of the cells, and the closest point will be the one at the centre of the cell.
Each edge is the perpendicular bisector between two neighbouring points, dividing the plane in two half planes, containing the set of points closest to each of them.
Construction of Voronoi diagrams can be done incrementally, but in order to obtain fast query times, one needs to decompose the cells into simpler structures.

2.5.4 *Delaunay Triangulation*

Another useful structure for geometric algorithms is the Delaunay triangulation. A Delaunay triangulation is a special kind of triangulation with many useful properties. In an unconstrained Delaunay triangulation, each triangle's circumcircle contains no points other inside its circumference.
It maximizes the minimum angle in its triangles, avoiding long or slender triangles. The set of all its edges contains both the minimum-spanning tree and the convex hull. The Delaunay triangulation is unique for a set of points, except when it contains a subset that can be placed along the same circumference.

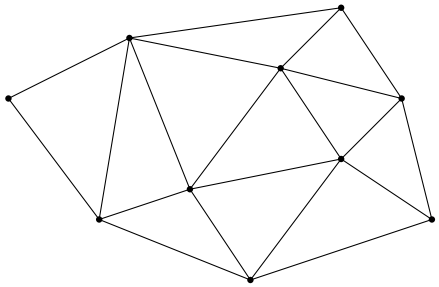

Figure 2: Delaunay Triangulation Example

More importantly, the Delaunay triangulation of a set of points is the dual graph of its Voronoi Diagram. The edges of the Voronoi diagram, are the line segments connecting the circumcentres of the Delaunay triangles. When overlapped, the duality becomes more obvious, with the Delaunay edges, in black, connect the points at the centre of the Voronoi cells, in red, to their neighbours:

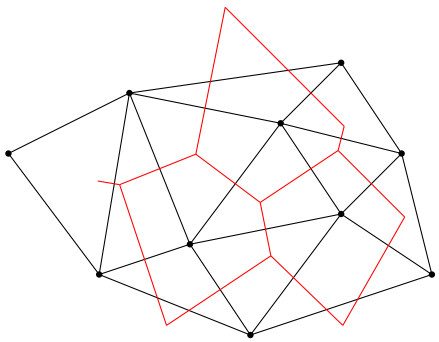

Figure 3: Voronoi Diagram and Delaunay Triangulation

Unlike its counterpart, the Delaunay is much simpler to build incrementally. It is also easier to work with, whilst still providing most

of the Voronoi diagram's properties, including the ability to calculate both point location and nearest neighbour searches.

2.5.4.1 *Greedy Routing*

In order to quickly calculate the nearest neighbour to a point in a set, one can make use of the Delaunay triangulation.
Consider a triangulation $\mathcal{T}$. In order to find the closest vertex in $\mathcal{T}$ to a new point p_{dst}, we must start at an arbitrary vertex of $\mathcal{T}$, v, and find a neighbour u of v whose distance to p_{dst} is smaller than the distance between p_{dst} and v. Repeat the process for u and its neighbours. When we reach a point w such that no neighbours of w are closer to p_{dst} than w is, we have found the closest point to p_{dst} in w.

Theorem 2.5.1. *There is no point set whose Delaunay triangulation defeats the greedy routing algorithm.*

Proof. For every vertex v in a triangulation $\mathcal{T}$, if there is at least one neighbour of v, u closer to p_{dst} than v is, let the perpendicular bisector of the line segment defined by v and u be called e. e intersects the line segment (v, p_{dst}) and divides the plane in two open half planes: h_v and h_u. h_u contains p_{dst}.
Delaunay edges connect the Voronoi neighbours and their bisectors define the edges of the Voronoi cells, which are convex polygons.
If we arrive at a point w, whose neighbourhood contains no points closer to p_{dst} than itself, then p_{dst} is contained within all possible open half planes containing w, defined by w and all its neighbours. p_{dst} is then by definition located in w's Voronoi cell. This means that w is the point in $\mathcal{T}$ closest to p_{dst}. □

2.5.4.2 *Construction*

2.5.4.3 *Deconstruction*

2.5.4.4 *Half-Edge Structure*

3

DETERMINISTIC ALGORITHMS

4

HEURISTIC ALGORITHMS AND FUTURE WORK

5

CONCLUSION