Visualisation and Analysis of Geographic Information: Algorithms and Data Structures

João Valença

December 9, 2014

ABSTRACT

Short summary of the contents of your thesis.

CONTENTS

1

DETERMINISTIC APPROACH

1.1 NAÏVE

1.2 INTEGER LINEAR PROGRAMMING

A different, and interesting approach to the problem would be to formulate it in linear programming, and use a solver to get the optimal set of centroids. To best describe our problem mathematically, our objective function can be formulated as:

$$\min_{\substack{C \subseteq P \\ |C|=k}} \max_{p \in P} \min_{c \in C} \|c - p\| \tag{1}$$

Where P is the initial set of points, C the set of centroids, and k the number of centroids given by the problem. This, however, is not easily or efficiently convertible to a linear programming language. To overcome these problems, we can compromise the optimal centroid attribution for each point. Since our objective function minimises the maximum distance of any point to its centroid, we can allow the points to be assigned to any centroid, so long as the maximum distance between a point and its centroid is smaller than or equal to the largest distance between a point and its centroid:

$$\min D \tag{2}$$

$$s.t. \sum_{i=1}^{N} y_i = k \tag{3}$$

$$s.t. C_{ij} x_{ij} \leq D \qquad \forall i, j \in [1, N] : i < j \tag{4}$$

$$s.t. \sum_{i=1}^{N} x_{ij} \geq 1 \qquad \forall j \in [1, N] \tag{5}$$

$$s.t. \sum_{j=1}^{N} x_{ij} \leq N \cdot y_i \qquad \forall i \in [1, N] \tag{6}$$

This formulation achieves the same values for the objective function, and a valid set of centroids for this value, but only the farthest point from its centroid will be guaranteed to be correctly assigned. In order to get the correct affectations, we need to iterate over all points and assign them to their respective centroids, which can be done in polinomial time.

1.3 INCREMENTAL BRANCH-AND BOUND

1.3.1 *Branching*

1.3.2 *Bounds*

1.4 DELAUNAY ASSISTED INCREMENTAL BRANCH-AND-BOUND

1.4.1 *Pre-processing*

1.4.2 *Branching*

1.4.3 *Centroids and Delaunay Triangulation*

Algorithm 1 Insert Centroid

```
1: procedure INSERT_CENTROID X
2:     Find Triangle Containing X$\mathcal{O}(\log n)$
3:     Update Delaunay triangulation $\mathcal{O}(n)$
4:     for N in Neighbours(X) do
5:         for i in CoveredBy(N) do
6:             if $dist(X) < dist(N)$ then
7:                 $CoveredBy(N) \leftarrow CoveredBy(N) \setminus i$
8:                 $CoveredBy(X) \leftarrow CoveredBy(X) \cap i$
9:                 Update Objective Function
10:            end if
11:        end for
12:    end for
13: end procedure
```

Algorithm 2 Remove Centroid

```
1: procedure REMOVE_CENTROID X
2:     for N do in neighbours(X)
3:         for i do in CoveredBy(X)
4:             if $N = argmin(dist(i, neighbour(X)))$ then
5:                 $CoveredBy(N) \leftarrow CoveredBy(N) \cap i$
6:                 $CoveredBy(X) \leftarrow CoveredBy(X) \setminus i$
7:                 Update Objective Function
8:             end if
9:         end for
10:    end for
11:    Remove X from Delaunay Triangulation $\mathcal{O}(n)$
12: end procedure
```

1.4.4 *Non-centroids and Greedy Routing*

Algorithm 3 Insert Non-Centroid

```
1: procedure INSERT_NON_CENTROID Y
2:     Pick good candidate for closest centroid (C)
3:     while $dist(Y, C) \neq \min(dist(Y, C \cup neighbour(C)))$ do
4:         $C \leftarrow argmin(dist(Y, C \cup neighbour(C)))$
5:     end while
6:     $CoveredBy(C) \leftarrow CoveredBy(C) \cap Y$
7:     Update Objective Function
8: end procedure
```

Algorithm 4 Remove Non-Centroid

```
1: procedure REMOVE_NON_CENTROID Y
2:     Pick closest centroid (C)
3:     $CoveredBy(C) \leftarrow CoveredBy(C) \setminus Y$
4:     Update Objective Function
5: end procedure
```

1.4.5 *Bounds*

1.5 RESULTS AND ANALYSIS

2

HEURISTIC APPROACH

3 CONCLUSION