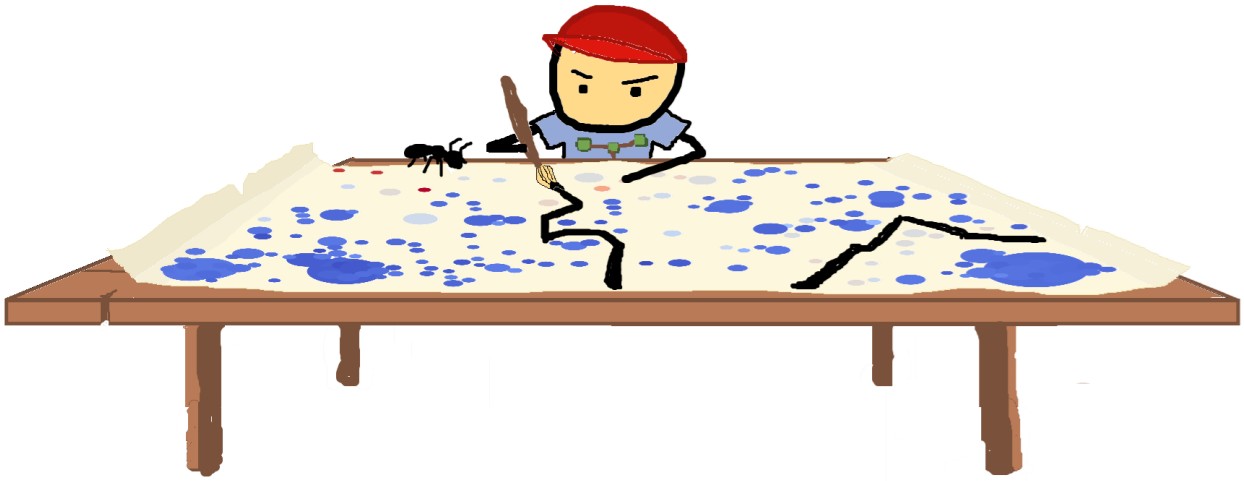


# Delivery times and Regionalization

## A study in unsupervised partitioning of lead time data through spatial clustering

Master's thesis in Complex Adaptive Systems in collaboration with Centiro Solutions AB

ARON BÖRJESSON LINDGREN

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

# Delivery times and Regionalization

A study in unsupervised partitioning of lead time data through spatial clustering

ARON BÖRJESSON LINDGREN


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2024

Delivery times and Regionalization
A study in unsupervised partitioning of lead time data through spatial clustering
ARON BÖRJESSON LINDGREN



Supervisors: Mikael Böörs & Joonas Halmkrona Lahtinen, Centiro Solutions AB
Examiner: Rebecka Jörnsten, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Delivery times and Regionalization
A study in unsupervised partitioning of lead time data through spacial clustering
ARON BÖRJESSON LINDGREN
Department of Mathematical Sciences
Chalmers University of Technology

# Abstract

This project aimed to find a solution to partition geographical maps into regions based on historical lead time data of delivered parcels. The guidelines of the partitioning were to keep lead time distributions of areas within one region as similar as possible, while keeping distributions of different lead times as different as possible. The partitioning algorithm would also preferably execute the task in reasonable time. Two algorithms were implemented and tested. One was an adaptation of Ant Colony Optimization, that proved to strongly respect the specified criteria, but at the cost of very high computation time. The second algorithm goes by the name REDCAP for spatial clustering, and this one was more thoroughly explored. It works by building a minimum spanning tree between all atomic level areas as nodes, clustering them together in a bottom-up approach while considering up to all nodes in every currently merged cluster along the way. The algorithm comes with some options to be specified: How to compute the dissimilarity between two clusters (single-linkage, complete-linkage, average-linkage), and which nodes of the clusters to consider (first-order constraint, full-order constraint). The data, represented as histogram vectors with each bin representing days passed since pickup, all split up by pickup day of the week, also had to be compared using some measure of dissimilarity. The dissimilarity measures explored were L1-distance, L2-distance, earth mover distance, and Jaccard index. This allowed for many different possible partitionings despite the same principle. To compare the result, consensus clustering was applied to measure how consistently the same algorithm partitions the same nodes of a data set together when making small adjustments to the data itself. It allowed to determine the quality of each option combination, and its ability to select how many regions to suggest. For a couple of data sets, single-linkage and earth-mover distance failed to give out a clear result and were removed as options. First-order constraint had a habit of cutting out single node size regions, and were therefore also removed. The combinations of the other options will all run on any new data set, and they each got a vote for the final partitioning. Some visual plots confirm that this approach seems to give reasonable results. Since deliveries tend to take longer to islands and remote locations, the algorithm tends to perform most of its partitioning there, and group country mainlands into one single region. Deliveries in Sweden showed to be an exception to this pattern, due to a lot of remoteness in the mainland, yielding regions of more equal area size. These regions were compared to those of the provided estimations of lead time, which turned out to match well overall, with some minor differences.

Keywords: Regionalization, Spatial clustering, Graph theory, REDCAP, ACO, Consensus clustering.

# Acknowledgements

I would like to express my sincere gratitude to everyone who has supported and guided me throughout this project. First and foremost, I extend my deepest thanks to my company supervisors, Mikael and Joonas, for their continuous attention and feedback, whether it involved brainstorming solutions, discussing results, or refining documentation. I am also profoundly grateful to my university supervisor and examiner, Rebecka Jörnsten, for her invaluable assistance, profound knowledge, and keen understanding of all aspects of the project. Lastly, I wish to thank my fellow master thesis colleagues, Wendy and Hampus, for their dedication and effort in following and understanding my project, as well as for helping me identify and correct numerous documentation errors.

Aron Börjesson Lindgren, Gothenburg, June 2024

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| SKATER | Spatial 'K'luster Analysis by Tree Edge Removal |
| REDCAP | Regionalization with Dynamically Constrained Agglomerative Clustering and Partitioning |
| MST | Minimum Spanning Tree |
| ACO | Ant Colony Optimization |
| EMD | Earth Mover Distance |
| AIC | Akaike Information Criterion |
| BIC | Bayesian Information Criterion |
| PMF | Probability Mass Function |
| CDF | Cumulative Distribution Function |
| PAC | Proportion of Ambiguous Clustering |

# Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

## Terminology

| | |
|---|---|
| *Node* | Equivalent to "location" where one or several parcels has been delivered |
| *Edge* | Connects two adjacent nodes or regions |
| *Ridge* | A border between two nodes. Every ridge crosses one and only one edge |
| *Vertex* | A point on the map where multiple ridges meet |

## Indices

| | |
|---|---|
| $u$,$v$ | Indices for nodes |
| $i$, $j$ | Indices for vertices |
| $i$ | Index for other sets |

## Sets

| | |
|---|---|
| $R$ | ”Region”; a set of connected nodes |
| $\mathbf{R}$ | The set of all regions |
| $S$ | The complete set involving all nodes |
| $G = (S, E)$ | The graph including the node set $S$ and a set $E$ of edges. |
| $G'$ | An undirected positional graph where every cycle encloses exactly one node in $S$ |
| $U, V, C$ | Connected components in $G$. |

## Parameters

| | |
|---|---|
| $k$ | Number of regions |
| $m$ | Lead time histogram vector length |

## Variables

| | |
|---|---|
| $N$ | Total number of nodes in $S$ |
| $M$ | Total number of edges in $E$ |
| $d^{(u,v)}$ | "Dissimilarity" between two nodes $u$ and $v$. |
| $n$ | Total amount of deliveries within the full set of nodes |
| $n_u$ | Amount of deliveries to node $u$ (area $z_u$) |
| $z_u$ | "location"; a geographical area represented by node $u$ |
| $\mathbf{T}^{(u)}$ | Lead time histogram vector/matrix for node $u$ |
| $\mathcal{I}_k$ | Relative increase in proportion of ambiguous clustering from $k-1$ to $k$ |

# Contents

**List of Acronyms** **ix**

**Nomenclature** **xi**

**List of Figures** **xv**

**1 Introduction** **1**
1.1 Background . . . 1
1.2 Aim . . . 1
1.3 Limitations . . . 2

**2 Theory** **3**
2.1 Different principles of partitioning a map . . . 3
2.1.1 SKATER . . . 5
2.1.2 REDCAP . . . 6
2.1.3 Ant Colony Optimization . . . 7
2.2 Measuring distribution similarity . . . 9
2.2.1 $L_1$- and $L_2$-distance . . . 10
2.2.2 Earth mover distance . . . 11
2.2.3 Jaccard index . . . 12
2.3 An objective function to quantify a solution . . . 13
2.3.1 Information criterion . . . 13
2.3.2 Consensus clustering . . . 14
2.3.2.1 Consensus clustering for merging solutions . . . 16

**3 Methodology** **17**
3.1 Converting the data into a graph . . . 17
3.1.1 Node placement . . . 17
3.1.2 Edges . . . 19
3.1.3 Representation of lead time data within nodes . . . 20
3.2 Custom generated data sets . . . 23
3.3 REDCAP implementation . . . 24
3.4 Ant colony partitioning . . . 26
3.5 Testing procedure . . . 29

**4 Results** **33**
4.1 REDCAP option elimination . . . 33

4.2 Comparison with carrier specified lead times . . . . . 41
4.3 Performance on other data sets . . . . . 42
4.4 ACO results . . . . . 43

**5 Discussion** **45**

**Bibliography** **49**

**A Sweden fooling S. Monti et al.'s method of determining $k$** **I**

**B Confusion matrices for data set 2, $k = 5$** **III**

**C Individual partitionings from 12 REDCAP setting combinations on data set 4** **V**

# List of Figures


2.1 A set of nodes merged into three clusters. . . . . 4
2.2 A set of nodes evenly distributed on a one-dimensional space. . . . . 4
2.3 Example of an MST from a set of nodes. . . . . 5
2.4 An illustration of First-Order constraining (red lines) and Full-Order constraining (red lines + blue dotted lines). . . . . 6
2.5 $U$ and $V$ with example values. . . . . 7
2.6 Histogram plot of two simple discrete distributions. . . . . 10
2.7 EMD flow graph for example vectors with edges and costs for $i = 3$ drawn. . . . . 12
2.8 Generated example nodes. . . . . 15
2.9 Two instances of $C^{(k)}$. . . . . 15
2.10 Consensus plot for a range of k-values. . . . . 15

3.1 Illustration of the H3 hexagons for some resolution 0 and 1, borrowed from [2]. . . . . 18
3.2 Germany set visualization with zip-code nodes. . . . . 19
3.3 Germany set after H3 grouping by resolution 4. . . . . 19
3.4 Voronoi diagram of data points in Germany after merge by H3 resolution 5. . . . . 20
3.5 Scatterplot of lead time over pickup time of the week. . . . . 21
3.6 Hourly bar graph of deliveries picked up on a Thursday. . . . . 21
3.7 Parcel pickup frequency over a year. . . . . 22
3.8 A simple illustration of Simpson's paradox. . . . . 22
3.9 A figure showing $\mathbf{T}^{(u)}$ for an example instance of $u$. . . . . 23
3.10 An instance of generated data with five distinct pre-defined regions. . 24
3.11 Lead time distributions split by intended region for the generated data set. . . . . 24
3.12 A second example of a Voronoi diagram. . . . . 26
3.13 An example of one ant partitioning of figure 3.12. . . . . 29

4.1 PAC over $k$ for a range of options for data set 1. . . . . 33
4.2 PAC over $k$ for a range of options for data set 2. . . . . 35
4.3 PAC over $k$ for a range of options for data set 3. . . . . 36
4.4 Six partitionings of data set 3 with different options. . . . . 37
4.5 Partitioning after merging suggestions from figure 4.4 by majority voting. . . . . 38
4.6 Co-association matrix from the six suggestions in figure 4.4. . . . . 38

4.7 PAC over $k$ for a range of options for the data set 4. . . . . 39
4.8 Six partitionings of data set 4 with different options. . . . . 40
4.9 Data set 4 partitioned after 3 votes. . . . . 41
4.10 Data set 4 partitioned after 6 votes. . . . . 41
4.11 Data set 4 partitioned after 12 votes. . . . . 41
4.12 Data set 5 with no grouping or filtration. . . . . 41
4.13 Carrier specifies estimations for data set 5. . . . . 41
4.14 Difference, representing how well deliveries in figure 4.12 were on time. 41
4.15 Partitioning of Europe. . . . . 42
4.16 Partitioning of Spain. . . . . 42
4.17 Partitioning of Germany . . . . . 43
4.18 Partitioning of UK. . . . . 43
4.19 Germany partitioned into 8 regions by REDCAP. . . . . 44
4.20 Germany partitioned into 8 regions by ants. . . . . 44

A.1 Delivery data for Sweden . . . . . I
A.2 Consensus plot for a range of k-values . . . . . I

# 1

# Introduction

## 1.1 Background

Whenever a parcel is out for delivery, it is always of relevance to have a time estimation for arrival. Not only is it important for maintaining the logistics, but also for customer experience. When it comes to last-mile logistics, parcels are picked up by a carrier from an origin such as a warehouse, store, etc., and transported to a destination like a collection point, private postal box, etc. Parcels come in different sizes, at different times of the day/week, and are transported to different destinations from different origins by different carriers. Even excluding the probability that any unforeseen event may delay the delivery, assigning a suiting maximum delivery time for every carrier for every parcel is certainly a challenge. Not only are the involved parameters many, but also of different types; some are scalars while others are categories, for example.

One parameter that does, not unsurprisingly, affect the delivery time is the location of the destination. Very often, it takes significantly longer time to receive your delivery if you live on a remote island, than if you live in the capital city, and which country you live in may affect the time as well. To involve location of destination is troublesome, however, when performing data scientific research. Parameterizing location as 2D-scalars of geographical coordinates may struggle to capture tendencies, and available categories like zip-codes or country are in some cases just too many to uncover spatial patterns. To involve the destination parameter in a fair way, the categorization needs to be done smarter, and on a larger scale.

## 1.2 Aim

This project aims to aid future data science research by uncovering a way to reduce the destination parameter into some larger scale grouping. The goal is to construct an algorithm that can partition a given geographical map into separate spatial regions based on existing delivery data and the following specifications:

1. The distribution of delivery lead times within one region is as homogeneous as possible.

2. The distributions of adjacent regions are as different as possible.

3. Every region is connected as one area, and doesn't overlap with another region.

Additionally, factors such as computation time and ability to handle varying data sets will be considered as well.

The number of requested regions are unknown beforehand. Any algorithm must decide on a number of regions based on the three given guidelines. The partitioning must be represented in a way that makes it possible to specify any position in geographical coordinates, or any geographical coordinates in a certain interval, and receive a region that this position belongs to.

A successful project would yield a better understanding of carrier performance with respect to location, which can be used as a factor in future delivery analysis, and ultimately help the supplier make better time estimations and strategical decisions in the future.

## 1.3 Limitations

Any testing will be performed using the data available, which covers deliveries in Europe throughout the year 2023. The information used for the task is country, zip code, time of departure and time of arrival. For one data set (see section 4.2) there will also be carrier specified estimations of lead-days available.

The project does not aim to conduct any research based on the concluded regions. It aims to construct the algorithm that performs the regionalization of any spatially distributed delivery data, with the gal that these regions will make it easier to include destination location in future research.

# 2

# Theory

This chapter is divided into three separate sections, representing three important pieces required to solve the problem. The first aims to select algorithms that serves as main principle towards a solution to the problem. The second section looks at possible ways to compare subsets of data. The third section investigates some possible ways of making unsupervised comparison of solutions possible, by quantifying a suggested solution into a single scalar value.

## 2.1 Different principles of partitioning a map

This section aims to answer the question what strategies are possible and what strategy, or combination of strategies, should qualify for implementation. There are numerous ways to approach the problem, each with their benefits and drawbacks. One may however group them up into three general principles:

- top-down slicing
- bottom-up clustering
- trial-and-error search and optimization

The first principle encapsulates strategies that start off by splitting the map into two regions, then selecting one of the current regions one at a time to split further until some satisfactory condition is met. The second principle works in the opposite direction. Here, nodes are merged together on an atomic level into clusters, effectively joining clusters until reaching a satisfactory condition, where the resulting joined clusters then represents the requested regions.

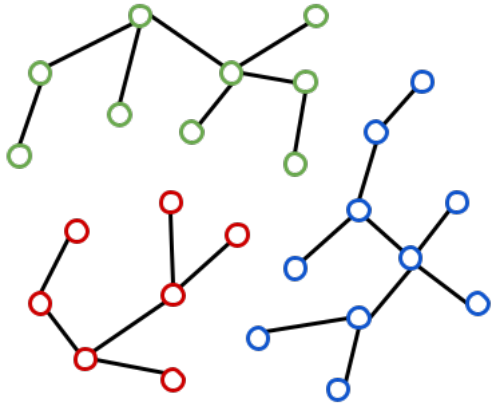


**Figure 2.1:** A set of nodes merged into three clusters.

These principles are both relatively fast, as every step ensures progress towards a result. However, this also means they may yield a suboptimal solution caused by an early split/merge when there was yet a long way until the final result would be visible. For example, consider a top-down slicing partitioning of the set of nodes in figure 2.2 distributed in a one-dimensional space.

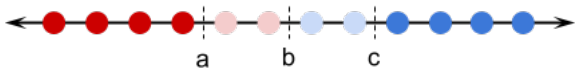



**Figure 2.2:** A set of nodes evenly distributed on a one-dimensional space.

The color of the nodes represents some value that ranges from blue to red, and the task is to separate the set into $k = 3$ regions. The top down slicing principle could in this scenario make the first split at $b$, since that seems to yield the two mostly different subsets at that step. Then, it would make a second split within one of these two regions, at either $a$ or $c$. However, in this scenario, making the two splits at $a$ and $c$ instead would be a more desirable result, as the distribution of color would then be more homogeneous within each region.

The third principle encapsulates all methods that starts off with some suggested partitioning, and then tries to progressively find better solutions by making adjustments. This could include guessing at random or based on successful partitions using genetic algorithms or similar, or tweaking existing borders using gradient descent (similarly to how a classification neural network is trained). These methods usually stand a better chance at finding optimal solutions, as they always have a chance to undo any error, but at the cost of much higher computation time, which makes them usually not feasible for larger data sets.

### 2.1.1 SKATER

The two most common algorithms that are mentioned when researching regionalization strategies in general, are the SKATER algorithm and it's extension called REDCAP. SKATER stands for Spatial 'K'luster Analysis by Tree Edge Removal and follows a hybrid principle of bottom-up and top-down, by splitting the algorithm into two steps that follows one principle each. At the first step, the algorithm takes the full adjacency graph $G$ with edges weighted by the dissimilarity $d$ between the nodes, and forms a so called minimum spanning tree (MST). An MST is in the Numpy and Scipy documentation defined as:

> A graph consisting of the subset of edges which together connect all connected nodes, while minimizing the total sum of weights on the edges.[10]

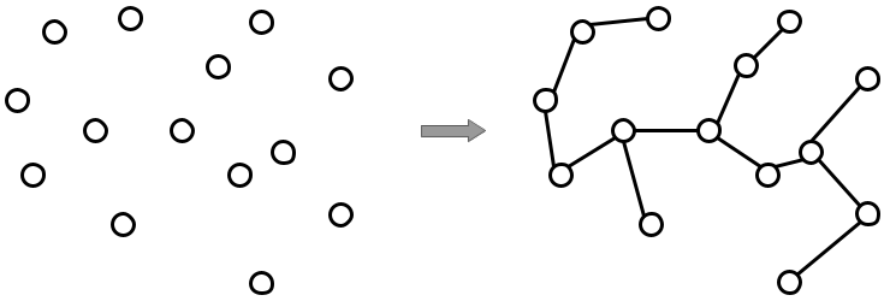


**Figure 2.3:** Example of an MST from a set of nodes.

The second step aims to remove a total of $k-1$ edges from the MST, so that $k$ connected components representing the $k$ regions remain. AssunÇãoFederal et. al. presents two approaches to this step. Both involve an objective function to quantify the outcome of each edge removal. The objective function they used goes by the name "sum of square deviations" (SSD) defined as

$$SSD_k = \sum_{j=1}^{m} \sum_{i=1}^{n_k} (x_{ij} - \bar{x_j})^2 \tag{2.1}$$

where $n_k$ is the number of spatial objects in tree $k$; $x_{ij}$ is the $j$th attribute of spatial object $i$; $m$ is the number of attributes considered in analysis; and $\bar{x_j}$ is the average value of the $j$th attribute for all objects in tree $k$.

The first approach simply iterates over all edges in the MST and evaluates how much the overall homogeneity improves in each connected component in that edge were to be removed. That is, if the tree $T$ is to be split up in two subtrees $T_a$ and $T_b$, the function evaluated is

$$f_1(S_I^T) = SSD_T - (SSD_{T_a} + SSD_{T_b}) \tag{2.2}$$

where $S_I^T$ is the arrangement produced by cutting out the edge $I$ from the tree $T$. This is repeated for all remaining connected components until $k-1$ edges are removed.

This is quite a computationally exhaustive approach, which motivates AssunÇãoFederal et. al's second option. This one searches for an optimum solution by only evaluating the neighbors of candidates already visited. This way the algorithm sort of "steps through" the tree in the direction that seems to improve the outcome of equation 2.2 until the best edge among its neighbors is found. This speeds up the algorithm from the first approach at the cost of the risk of reaching a suboptimal solution by getting stuck in a local maximum[12].

### 2.1.2 REDCAP

REDCAP is another regionalization algorithm inspired by, and therefore similar to SKATER. It also approaches the problem by building an MST from the set of nodes, but instead of carefully computing which edge is optimal to remove as a second step, it searches for an optimum already in the merging step. In fact, it comes with a total of six different possible options, three agglomerative clustering methods times two spatial constraining methods. The two constraining methods go by the names *First-Order constraining* and *Full-Order constraining*, and are illustrated in figure 2.4 below

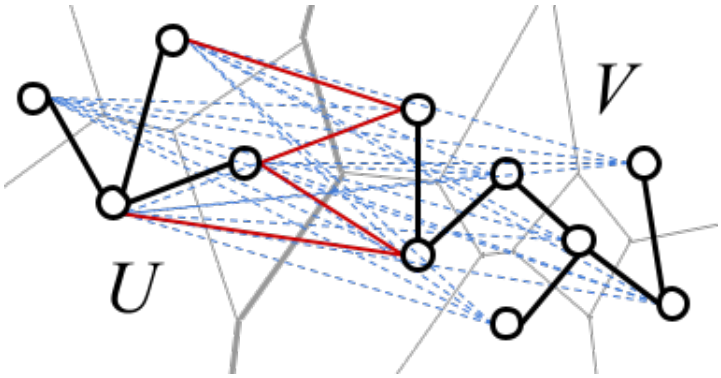



**Figure 2.4:** An illustration of First-Order constraining (red lines) and Full-Order constraining (red lines + blue dotted lines).

The connected components $U$ and $V$ are separated by the thickened gray border. When suggesting a merge of $U$ and $V$, First-Order constraining only considers the edges of adjacent nodes (marked in red), while Full-Order constraining considers the edges between any node in $U$ and any node in $V$ (the red lines and the blue dotted lines)[4].

To determine which exact edge defines the dissimilarity between $U$ and $V$, the linkage type comes in play. The three different linkage options of REDCAP are called *single-linkage*, *complete-linkage* and *average-linkage*. To illustrate how they work, consider figure 2.5 below, of the same examples of $U$ and $V$, but where every node has an example instance value displayed as a small blue digit. The single nodes $u \in U$ and $v \in V$ have been selected, and we want to determine the dissimilarity $d(U, V)$ between them following Full-Order constraining.

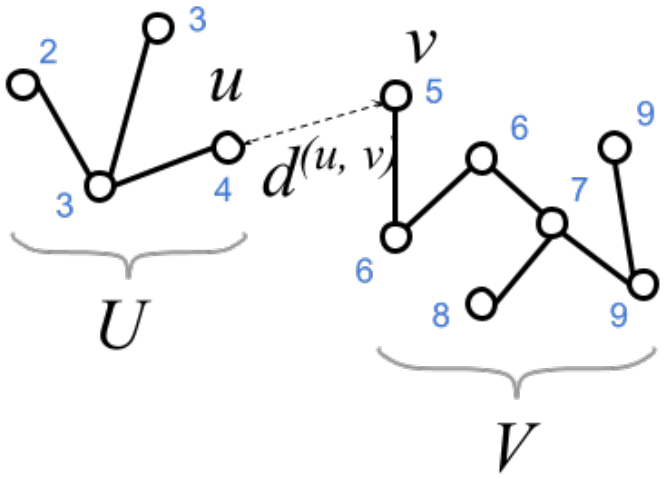



**Figure 2.5:** $U$ and $V$ with example values.

The three linkage types are defined as follows:[4]

$$\begin{aligned} \text{single-linkage: } d_{SLK}(U,V) &= \min_{u' \in U,\ v' \in V}(d^{(u',v')}) \\ \text{complete-linkage: } d_{CLK}(U,V) &= \max_{u' \in U,\ v' \in V}(d^{(u',v')}) \\ \text{average-linkage: } d_{ALK}(U,V) &= \frac{1}{|U||V|}\sum_{u' \in U}\sum_{v' \in V} d^{(u',v')} \end{aligned} \tag{2.3}$$

Suggesting that $d^{(u,v)}$ is the difference between their values, we would for these examples of $U, V$ get

$$\begin{aligned} d_{SLK}(U,V) &= |4-5| = 1 \\ d_{CLK}(U,V) &= |2-9| = 7 \\ d_{ALK}(U,V) &= 4.14 \end{aligned} \tag{2.4}$$

Note that the actual nodes contain more data than single values (this is further discussed in section 3.1.3) and the dissimilarity $d^{(u,v)}$ rather than being the difference between two values, is discussed in section 2.2.

The representation of single-linkage and complete-linkage for Full-Order constraining can be motivated as: single-linkage suggests two connected components $U, V$ are similar if any pair of nodes $u \in U,\ v \in V$ are similar, complete-linkage suggests two connected components $U, V$ are similar only if all nodes in $U$ are similar to all nodes in $V$.

### 2.1.3 Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms are inspired by how ants cooperate to find and bring home food. By secreting substances known as *pheromones* they leave a trace for other ants to follow. These pheromones are volatile hydrocarbons, which

evaporates over time. Thus, a path must be reinforced continuously to stay. Shorter paths to food have denser traversal than longer paths, which therefore increases their amount of pheromone. This further attracts more ants that will add more pheromone and eventually the ants will have established a path. One may simulate this behavior by having artificial ants traverse a graph. Every edge $e_{ij}$ holds a variable $\tau_{ij}$ that tells the current amount of pheromone, as well as a second variable $\eta_{ij}$ that holds some attribute of $e_{ij}$ that guides the ants towards an optimal solution faster. At every node, every ant needs to select an edge to traverse next. The probability that the ant at node $i$ selects the edge $e_{ij}$ is commonly calculated as

$$p(e_{ij}|S') = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{v_l \notin S} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}, \tag{2.5}$$

where $S'$ is the ordered set of visited nodes, and $\alpha, \beta$ are constants that tells how much the ants should listen to the values of $\tau$ and $\eta$ respectively.

A typical example to where ACO can be applied is the so called Traveling salesman problem (TSP). In TSP, a set of spatially distributed nodes is provided, and the goal is to find a closed route that visits every node exactly once, minimizing the total distance traversed. In this scenario, it would make sense to define the variables $\eta_{ij}$ inversely proportional to the Euclidean distance between node $i$ and node $j$. The example algorithm 1 below implements ACO to solve TSP.

**1.** Initialize pheromone levels:

$$\tau_{ij} = \tau_0, \ \forall i, j \in [1, N]\,. \tag{2.6}$$

**while** *Satisfactory solution not found* **do**

**2.** For each ant $k$, select a random starting node, and add it to the (initially empty) tabu list $L_T$. Next, build the tour $S$. In each step of the tour, select the move from node $j$ to node $i$ with probability $p(e_{ij}|S)$ given by equation 2.5. In the final step, return to the node of origin, i.e. the first element in $L_T$. Finally, compute and store the length $D_k$ of the tour.

**3.** Update pheromone levels:

**3.1** For each ant $k$, determine $\Delta\tau_{ij}^{(k)}$ as:

$$\Delta\tau_{ij}^{(k)} = \begin{cases} \dfrac{1}{D_k} \text{ if ant } k \text{ traversed edge} e_{ij}, \\ 0 \text{ otherwise.} \end{cases} \tag{2.7}$$

**3.2** Sum the $\Delta\tau_{ij}^{(k)}$ to generate $\Delta\tau_{ij}$:

$$\Delta\tau_{ij} = \sum_{k=1}^{n} \Delta\tau_{ij}^{(k)}. \tag{2.8}$$

**3.3.** Modify $\tau_{ij}$:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}. \tag{2.9}$$

**end**

**Algorithm 1:** An algorithm that approximates a solution to TSP using ACO, borrowed from [8] (Page 107).

In this algorithm, $n$ refers to the total number of ants, $k$ acts as an index for the ants, $D_k$ is the total distance traveled by ant $k$, and $\rho$ represents an evaporation rate of the pheromone.

## 2.2 Measuring distribution similarity

The variable $d^{(u,v)}$ was introduced in section 2.1.1 as a variable that quantifies the dissimilarity between the data of two nodes $u$ and $v$. There are many metrics to use to compute this value, with different levels of efficiency or bias; this chapter aims to list some candidates to use for this problem.

In chapter 3.1.3, we will motivate the choice of representing the data of each node as a so called *histogram vector* $\mathbf{T}_u$ for node $u$. A histogram vector can be defined as

> ... a mapping from a set of d-dimensional integer vectors i to the set of non-negative integers. These vectors typically represent bins (or their centers) in a fixed-size partitioning of the relevant region of the underlying space, and the associated integers are a measure of the mass of the

distribution that falls into the corresponding bin. [13] (Page 60)

A very simple general example of two histogram distributions is displayed in figure 2.6 below. The blue and red histograms both have 11 elements each, that are distributed slightly differently over a total of 6 bins. We'll name the blue one $\mathbf{T}^{(B)} = \left\{T_1^{(B)}, \ ..., \ T_6^{(B)}\right\} = \{1, 3, 4, 1, 2, 0\}$ and the red one $\mathbf{T}^{(R)} = \left\{T_1^{(R)}, \ ..., \ T_6^{(R)}\right\} = \{1, 2, 3, 3, 1, 1\}$. These will be referenced as example for the following ways to compute a dissimilarity score.

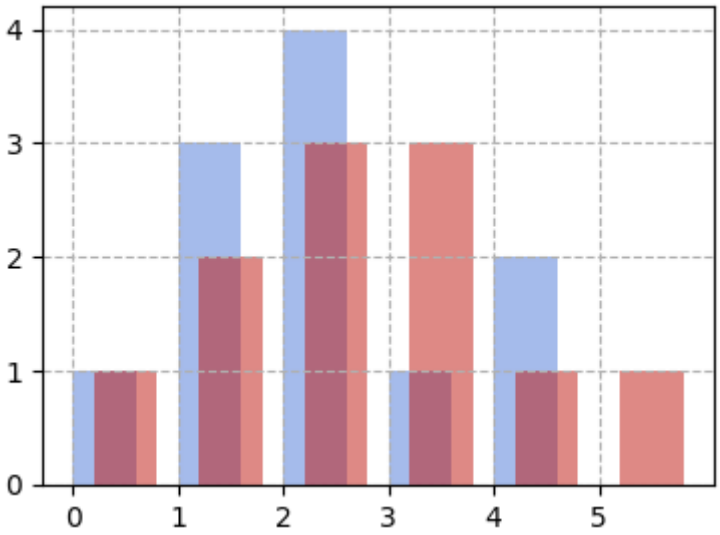



**Figure 2.6:** Histogram plot of two simple discrete distributions.

### 2.2.1 $L_1$- and $L_2$-distance

Two common metrics for comparing histograms are the so called $L_1$-distance and $L_2$-distance. These are both fast and simple to compute. They work by summing together the differences for each bin, or the squared differences for $L_2$. Formally, for histogram vectors $H^{(I)}, \ H^{(I')}$. [11]

$$\begin{aligned} d_{L_1}\left(H^{(I)}, \ H^{(I')}\right) &= \sum_{j=1}^{m} |H_j^{(I)} - H_j^{(I')}| \\ d_{L_2}\left(H^{(I)}, \ H^{(I')}\right) &= \sum_{j=1}^{m} \left(H_j^{(I)} - H_j^{(I')}\right)^2 \end{aligned} \tag{2.10}$$

Compared to $L_1$, $L_2$ puts emphasis on larger differences, which makes it more robust towards slightly noisy data, at the cost of being less robust towards unforseen events that may cause a shift or larger hole in the data. For the two example vectors $\mathbf{T}^{(B)}$

and $\mathbf{T}^{(R)}$ from earlier, the $L_1$- and $L_2$-distances would compute to

$$\begin{aligned} d_{L_1}\left(\mathbf{T}^{(B)},\ \mathbf{T}^{(R)}\right) &= 0+1+1+2+1+1=6 \\ d_{L_2}\left(\mathbf{T}^{(B)},\ \mathbf{T}^{(R)}\right) &= 0^2+1^2+1^2+2^2+1^2+1^2=8 \end{aligned} \tag{2.11}$$

### 2.2.2 Earth mover distance

The earth mover distance (EMD) metric refers to the distance between two distributions as the "minimal amount of work that must be performed to transform one distribution into the other by moving 'distribution mass' around" [13] (Page 59).

EMD is executed by generating a flow graph, where all bins are nodes and there are directed edges from all nodes of one distributions to all nodes of the other. The edges are directed from the distribution that gives earth, the *supplier*, to the distribution that received earth, the *consumer*. If the supplier doesn't have enough earth to fill the demands of the consumer, the problem can still be approached by simply swapping the roles. By letting $\mathcal{I}$ be a set of suppliers, $\mathcal{J}$ a set of consumers and $c_{ij}$ the cost to ship a unit of supply from $i \in \mathcal{I}$ to $j \in \mathcal{J}$, $d_{EMD}$ can be defined as the flow $f_{ij}$ that minimizes

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}f_{ij}, \tag{2.12}$$

subject to the constraints

$$\begin{aligned} f_{ij} > 0 \quad & i \in \mathcal{I},\ j \in \mathcal{J} \\ \sum_{i\in\mathcal{I}} f_{ij} = y_j \quad & j \in \mathcal{J} \\ \sum_{j\in\mathcal{J}} f_{ij} \le x_i \quad & i \in \mathcal{I}, \end{aligned} \tag{2.13}$$

where $x_i$ is the total supply of supplier $i$ and $y_i$ is the total capacity of consumer $j$. That is, for an optimal flow $\mathbf{F}$,

$$\mathrm{EMD}(x,y) = \frac{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}f_{ij}}{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} f_{ij}} = \frac{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}f_{ij}}{\sum_{j\in\mathcal{J}} y_j} \tag{2.14}$$

The denominator is a normalization factor that avoids favoring distributions of smaller size.[13] Looking at the red and blue example vectors again, and suggesting that the blue distribution is the supplier and the red distribution the consumer, the flow graph would look like as illustrated in figure 2.7.

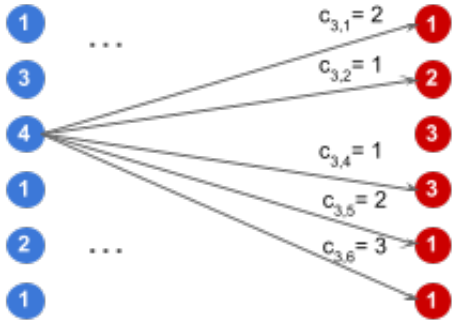



**Figure 2.7:** EMD flow graph for example vectors with edges and costs for $i = 3$ drawn.

The cost for any edge

$$c_{ij} = |i - j| \tag{2.15}$$

as that is how many steps any unit would need to move from one bin to another. Computing the EMD-value between $\mathbf{T}^{(B)}$ and $\mathbf{T}^{(R)}$ would yield the value 4. A minimum amount of flow can be achieved by moving one unit from $\mathbf{T}^{(B)}$ bin 1 two steps to $\mathbf{T}^{(R)}$ bin 3, one from bin 2 to 3 and one from bin 4 to 5.

The benefits with using EMD are that it is very adaptive, in the sense that it provides a fair metric even when comparing nodes $u, v$ where $n_u \neq n_v$, when bins are different, or when the lead times of $u$ and $v$ follows different distributions. However, in this application, the dissimilarity value $d^{(u,v)}$ will be calculated for a large number of nodes, and setting up and solving a graph problem for every $d^{(u,v)}$ will lead to long computation time in total.

### 2.2.3 Jaccard index

Jaccard index is a similarity measure with a lot of practical usage, as it is not only meant to compare statistical distributions, but any sets of data that can overlap. Assuming two potentially overlapping sets $A$ and $B$, the Jaccard index is defined as the relative overlap

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.16}$$

The dissimilarity version of the Jaccard index is therefore[7]

$$J_\delta(A, B) = 1 - J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \Delta B|}{|A \cup B|} \tag{2.17}$$

For the case of a probability distribution, one can interpret the distributions as overlapping surfaces. The Jaccard index would describe the relative overlap between

the probability density functions. For the two example distributions $\mathbf{T}^{(B)}$ and $\mathbf{T}^{(R)}$ from figure 2.6, the Jaccard index would be

$$J(\mathbf{T}^{(B)}, \mathbf{T}^{(R)}) = \frac{\sum_{i=1}^{m} min\left\{T_i^{(B)}, T_i^{(R)}\right\}}{\sum_{i=1}^{m} max\left\{T_i^{(B)}, T_i^{(R)}\right\}} = \frac{1+2+3+1+1+0}{1+2+4+3+2+1} = 0.62 \quad (2.18)$$

## 2.3 An objective function to quantify a solution

For any stochastic approach to be relevant, there must be a quantifiable way to measure the quality of a suggested region partitioning. Even for REDCAP, the number of regions $k$ is so far just a blind parameter, so to determine the optimal $k$, multiple values would have to be tested and evaluated. We are therefore interested in some function or algorithm that takes a suggested partitioning of $S$ into $k$ regions and returns a scalar value that reflects the quality of the partitioning.

### 2.3.1 Information criterion

On the topic selecting a model to represent some given data, there are two commonly mentioned functions that are often used to quantify the result; The Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC).

Assuming a model $M = \left\{\mathcal{L}_{\vec{\theta}} \mid \vec{\theta}\right\}$, where $\vec{\theta}$ is a vector of parameters required to describe the model, and $\mathcal{L}_{\vec{\theta}}$ is the likelihood of the model based on given $\vec{\theta}$ and observed data, AIC is expressed as[14]

$$\text{AIC}(M) = -2\ log\ \mathcal{L}_{\vec{\theta}} + 2|\theta| \quad (2.19)$$

The aim is to find a model and a set of parameters that minimizes AIC. $|\theta|$, the length of the parameter vector $\vec{\theta}$ is positive to penalize models that are too complex. An increased number of parameters usually improves the accuracy of the model, and therefore also the likelihood, but to avoid overfitting the model, one is rather interested in the best trade-off between model simplicity and likelihood.

The BIC is similar to AIC, but here we also criticise the number of data points $n$ in the observation. [14]

$$\text{BIC}(M) = 2\ log\ \mathcal{L}_{\vec{\theta}} - |\theta|\ log(n) \quad (2.20)$$

For this application, the $\mathcal{L}_{\vec{\theta}}$ term would represent a likelihood that describes how well all individual nodes belong to their assigned region. To calculate this, we use a tool called *multinominal joint probability mass function*. It calculates the probability that a number of $n$ samples are distributed exactly like in a specified histogram vector $\mathbf{T} = n_1, ..., n_k$ if drawn from a given distribution $\mathbf{p} = p_1, ..., p_k$. Given $\mathbf{T}, \mathbf{p}$ of length $k$, this probability is computed as[1]

$$P(X_1 = n_1, ..., X_k = n_k) = \frac{n!}{n_1!n_2!...n_k!} \cdot p_1^{n_1} p_2^{n_2} \dots \ p_k^{n_k}. \tag{2.21}$$

### 2.3.2 Consensus clustering

Another approach that is more closely related to the problem is to utilize *consensus clustering*. It's a way to evaluate the robustness of a partitioning $\boldsymbol{R}$ by measuring consensus across multiple runs of the partitioning algorithm. By applying consensus clustering to the result of a range of different values on $k$, the optimal $k$ can be selected as either the one that yielded the very best robustness or the highest $k$ before robustness decreases distinctively.

Consensus clustering proceeds as follows:

**Input:**

$S$: The full set of nodes
$k_{\min}, k_{\max}$: A range of $k$-values to explore
$H$: Number of resampling iterations
$b$: Number of nodes to block per iteration

**for** $k = k_{min}$ **to** $k_{max}$ **do**
  **for** $h = 1$ **to** $H$ **do**
    **1.** Define $S^{(h)}$ as a random subset of $S$ containing $N - b$ nodes.
    **2.** Run the partitioning algorithm on $S^{(h)}$ to generate the region set $\boldsymbol{R}^{(h)}$
    **3.** Define the matrix $\mathbf{M}^{(h)}$ of size $N \times N$ where $M_{uv}^{(h)} = 1$ if nodes $u$ and $v$ were assigned to the same region $R_r \in \boldsymbol{R}^{(h)}$, and 0 otherwise
    **4.** Define the matrix $\mathbf{I}^{(h)}$ of size $N \times N$ where $I_{ij}^{(h)} = 1$ if $i \in S^{(h)} \wedge j \in S^{(h)}$, and 0 otherwise
  **end**
  **5.** Compute the matrix $C^{(k)}$ where every element

$$C_{ij}^{(k)} = \frac{\sum_{h=1}^{H} M_{ij}^{(h)}}{\sum_{h=1}^{H} I_{ij}^{(h)}} \tag{2.22}$$

**end**

**Algorithm 2:** The consensus clustering algorithm.

$C^{(k)}$ now represents a consensus matrix unique to one specific $k$-value. Every element $C_{ij}^{(k)}$ ranges between $[0, 1]$ and represents how often nodes $i$ and $j$ were grouped together, where 0 means never grouped together and 1 means always grouped together.[9] A perfect consensus matrix therefore consists of only ones and zeros. Figure 2.9 below displays $C^{(k)}$ for $k = 4$ and $k = 5$ for the fictional set of nodes displayed in figure 2.8. The color represents the values of the elements from 0 (white) to 1 (red). The five distinct subsets are maintained throughout the consensus test, which yields the distinct consensus matrix for $k = 5$. For k=4 however, the algorithm is unsure about which subsets to group together, which causes fractional

values.

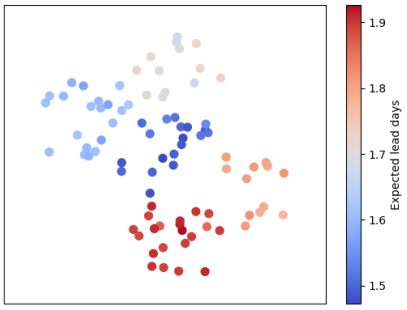



**Figure 2.8:** Generated example nodes.

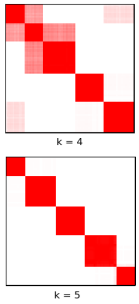



**Figure 2.9:** Two instances of $C^{(k)}$.

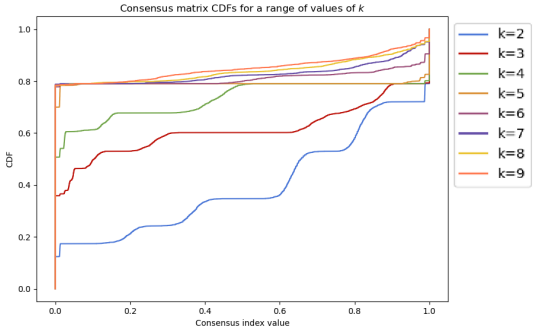



**Figure 2.10:** Consensus plot for a range of k-values.

Figure 2.10 displays the proportion of how much of each value in $[0, 1]$ is contained in each $C^{(k)}$. As $C^{(5)}$ only contains ones and zeros, its curve is completely flat. For $k < 5$, the curves are a bit uncertain about which clusters to group together causing some chunks of uncertain values. For $k > 5$ the algorithm starts to split up individual clusters, with very low consistency, causing an increase in values close to 1.0.

Based on this information, we may determine which value of $k$ seems most appropriate. S. Monti et al. suggests that

> The selection of the appropriate number of clusters proceeds by inspection of the CDFs' shape and progression as k increases. ... The inspection of the CDF progression is to select the largest k that induces a large enough increase in the area under the corresponding CDF. [9] (Page 102)

This approach would give the correct outcome when used on the example set above. However, other studies have pointed out that this approach doesn't always work. (A simple example that fools Monti et al.'s strategy can be found in appendix A). Y. Şenbabaoğlu et al. for example, suggest an optional approach. The Proportion of ambiguous clustering (PAC) is defined as

> ... the fraction of sample pairs with consensus index values falling in the intermediate sub-interval $(x_1, x_2) \in [0, 1]$. $x_1$ and $x_2$ are data-dependent thresholds, but will generally be chosen near 0 and 1 respectively. ... A low value of PAC indicates a flat middle segment, allowing inference of the optimal k by the lowest PAC. [15]

Finding appropriate values of $x_1$ and $x_2$ is a task in itself. The sub-interval should

catch the small misplacements for low k-values. If the interval $(w_1, x_2)$ is too small and all curves up until $k_{true}$ appear completely flat. One way to get around this, to avoid looking at the data to set appropriate $(w_1, x_2)$, would be to use a hybrid of the two methods suggested so far. Instead of observing $\Delta k$, one may observe $\Delta$PAC, or even, to avoid getting fooled by data sets where fluctuations after true $k$ are large, this self defined relative increase

$$\mathcal{I}_k = \frac{\mathrm{PAC}_k - \mathrm{PAC}_{k-1}}{\mathrm{PAC}_{k-1}}. \tag{2.23}$$

The optimal $k$ would then be the largest $k$ before $\mathcal{I}_k$ is large, i.e. PAC starts to increase. An example of this can be found in appendix A.

#### 2.3.2.1 Consensus clustering for merging solutions

The theory of consensus clustering can also be applied when different algorithms, or one algorithm with different parameters, yield different results. Previously, the confusion matrices $C^{(k)}$ described how often each pair of nodes were grouped together over an $H$ amount of repeated solving for slight variations of the data set. The confusion matrix, or in this case *co-association matrix*, can be generated in a similar manner using different partitionings of the same data set. Then, this co-association matrix may be the base for a final decision of clusters. This is commonly used when clustering data in high-dimensional space, as a consensus merge of the data projected on multiple low-dimensional sub-spaces is usually faster, and may even produce a better result than performing the clustering algorithm in all dimensions at once[3].

There are a couple of different ways to determine the final clusters based on the co-association matrix, each depending on the data and the problem in question. The simplest method is to use majority voting. That is, any two nodes $u, v$ belong to the same region only if $C_{uv} >= 0.5$. C. X. Gao et. al. state that one must however be careful about this method, as when the approaches are very diverse or generated from a method such as random-k, the combined result could risk being worse than all individual ones[3].

# 3

# Methodology

In this chapter, we will begin by concluding the format of the data we are working with, and then proceed to how REDCAP and ACO was applied to solve the regionalization problem.

## 3.1 Converting the data into a graph

A *graph* is a set of nodes connected by a set of edges, and it is very common within maths and algorithm design to attempt to represent a problem with a graph. It can make it both easier to understand how to derive a solution, and there's a lot of algorithms for graphs existing already, and the problem of partitioning a positional dataset into regions is no exception.

### 3.1.1 Node placement

To represent the problem using graph theory, we'll need to define the nodes and what information they contain. The source data provides a long list of deliveries, including information about when and where the parcels were picked up and delivered. A naive thought would be to have one node per delivery, at the location it was delivered to. However, since the aim of the project is to find regions where the distribution of lead times are as different as possible between regions, the deliveries would at some point need to be grouped together into subsets that collect all their lead time distributions we compare. For a trial-and-error approach, this grouping could possibly wait until any suggested region borders will be evaluated. However, this isn't a feasible approach for a couple of important reasons. Firstly, delivery location isn't available in the data at a resolution higher than zip-codes. This already means that deliveries would at least need to be grouped together on zip-code level, but the following three reasons motivate why it would be beneficial to aim for an even lower resolution level:

1. Grouping $n$ deliveries as a middle step of any algorithm would give a very high time complexity as $n$ is usually very large.

2. Many approaches (especially bottom-up) are mostly blind towards the final

result in the beginning of the algorithm, and therefore performs worse the less information that is visible in the initial merging steps.

3. For most data sets, we aren't interested in region borders that carefully cruises through cities or sparsely inhabited countrysides. Such borders are simply more fine than necessary.

To reduce the resolution level and the total amount of nodes $N$, we therefore utilize the *H3-framework*. H3 was developed by Uber, to easily display where Uber requests were denser. It partitions the earth into pre-defined hexagons. Each hexagon encapsulates seven smaller hexagons (see figure 3.1), that are at one resolution level higher.[2] By specifying a point on a geographical map and a resolution level, the H3-framework can return an index to the hexagon at that resolution that contains that point. The resulting node after grouping up all zip-code locations within one hexagon gets the coordinates of the hexagon's center point. This way, multiple nodes in a small area can be grouped together, joining their sets of delivery data, improving both $N$ and the average $n_u$.

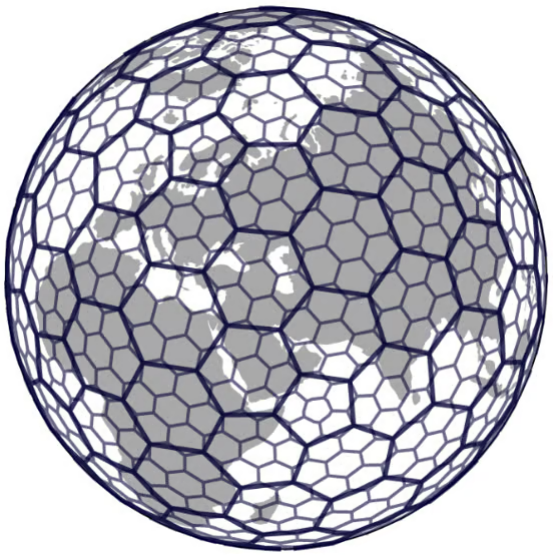


**Figure 3.1:** Illustration of the H3 hexagons for some resolution 0 and 1, borrowed from [2].

Figure 3.2 below displays a visualization of deliveries in Germany, where deliveries are grouped together into one node per zip-code, and 3.3 below displays the result after further grouping the nodes by hexagon resolution 4.

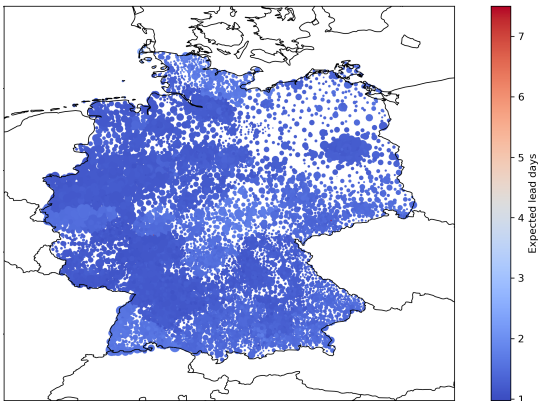



**Figure 3.2:** Germany set visualization with zip-code nodes.

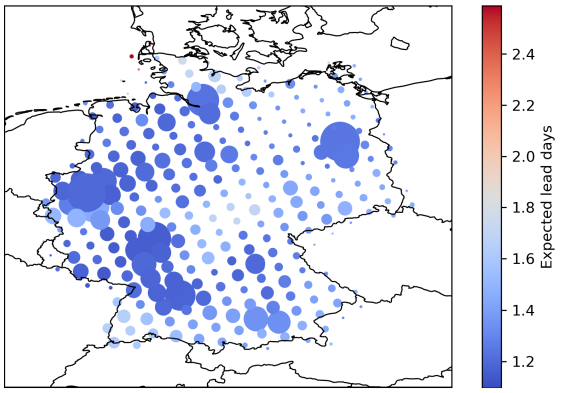



**Figure 3.3:** Germany set after H3 grouping by resolution 4.

The colors of the nodes represent the averaged lead times within each node, from cooler colors (shorter lead times) to warmer (longer lead times), and the sizes of the nodes tell how many deliveries are included in each node. This applies to every data set visualization throughout the report. (The span of both color and size are however relative to the values in the visualized data set, and are therefore not equal between every plot.)

### 3.1.2 Edges

The edges of a graph marks which nodes are connected to each other. For this problem, we'd like to put edges only between the nodes that are considered adjacent, to ensure we fulfill the third specification given in section 1.2. If there exist a delivery record to all hexagons at a specified resolution and area, the edges of the graph would simply be those that connect adjacent hexagons. Then, all nodes would be adjacent to at most six other nodes. However, in some cases there might be areas with a lacking data record, or simply uninhabited (for example, areas covered by water). To avoid including nodes with $n_u = 0$, the adjacency between nodes will be defined slightly differently.

`Scipy.spatial.Voronoi` is another framework that can return a so called *Voronoi Diagram* from a set of $N$ points in a plane (and in higher dimensions as well). A Voronoi diagram "subdivides the plane in exactly $N$ cells enclosing the portion of the plane that is the closest to each point"[5]. For areas where every hexagon contains a node, the Voronoi cells will resemble the hexagon shape (since the hexagons gets slightly skewed when projected on a 2D-plane, and the Voronoi cells are computed based on the skewed center points of the hexagons, the resulting Voronoi cells will for most of Europe approach the shape of squares instead. The cells do however maintain six sides, which means the projection skewing doesn't affect the validity of this tool). In those areas that are empty, the adjacent cells instead extend to cover the empty area. This effect can be seen in for example the sparsely inhabited north-east Germany (see figure 3.4).

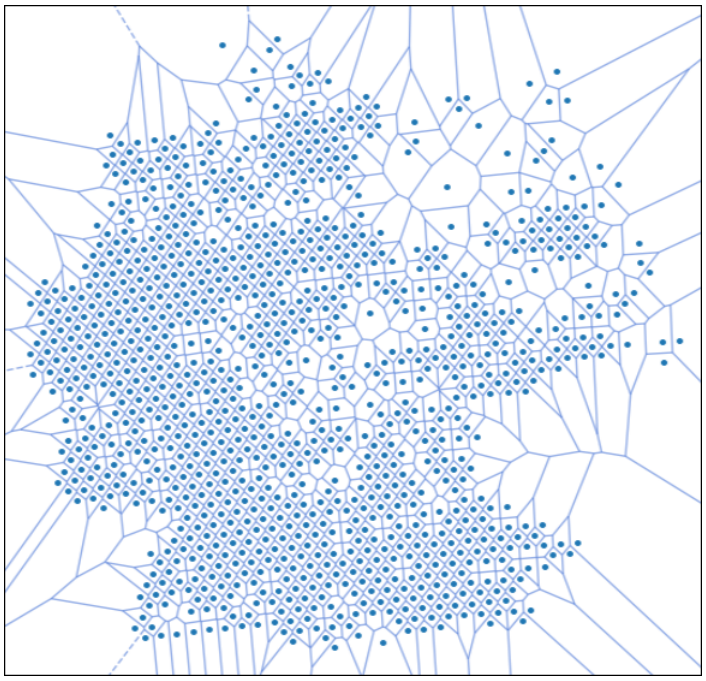


**Figure 3.4:** Voronoi diagram of data points in Germany after merge by H3 resolution 5.

The convenience of this tool opens up a second option to reduce the number of nodes in a data set. By simply filtering away nodes with fewer deliveries than a given threshold, and letting `Scipy.spatial.Voronoi` define the area borders where these nodes were positioned (like the case in figure 3.4), not only does the computation time go down, but it also reduces the risk of getting one-region nodes as an effect of large dissimilarities caused by insufficient data within some nodes. One may however keep in mind that this approach of generalizing regions where data is insufficient may not be optimal for applications where partitioning areas with low data history is important for the application.

### 3.1.3 Representation of lead time data within nodes

Thanks to the H3-framework, we now have a set of lowest-level nodes that gather all deliveries delivered to any point in the corresponding hexagonal area. What's left is to decide on a way to contract all $n_u$ deliveries of every node $u$ in some way that reduces the amount of required operations for distribution comparison, from $O(n)$ to something preferably much smaller.

One strategy would be to parameterise the data. By assuming the data follows some distribution function, one could fit that function to the data to estimate some minimum required parameters for reproducing a similarly distributed set of data. The main problem with this however, is that the differences between the fit distribution and the original data are often larger than the differences between the distributions of the nodes, which is what we intended to compare. For example, figure 3.6 below

shows the densities of lead times for parcels picked up on a Thursday. The density distribution seems to follow a combined shape of a gamma distribution and a sinus function. The gamma distribution tells the tendencies of lead times, and the sinus tells that deliveries occur more often in daytime than at night. However, the third wave appears to be missing. It is also visible in figure 3.5, as white wider periodic lines of significantly less data. This is easily explained by our society's standards not to work, and therefore not to deliver on weekends. However, a parameterized model would need to be quite complex in order to accurately capture this distribution shape.

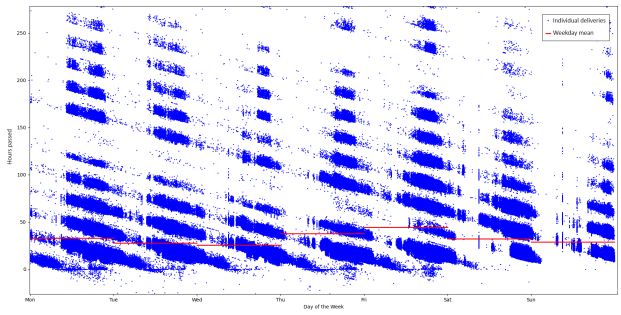



**Figure 3.5:** Scatterplot of lead time over pickup time of the week.

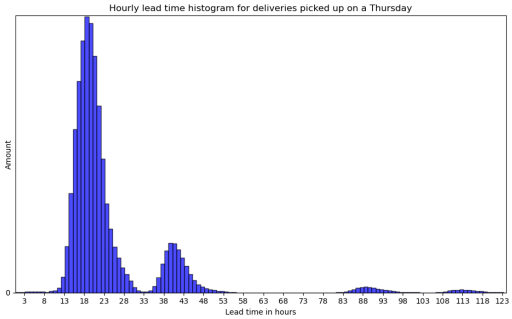



**Figure 3.6:** Hourly bar graph of deliveries picked up on a Thursday.

The sinus pattern representing day-night periods is both intuitive and consistent. In fact, this pattern makes the analysis more complicated, may disturb some measurements, and is not very interesting for the topic in question. Just like we reduced the resolution of positional accuracy in section 3.1.1, it would make sense to do it once again, but on the time axis. By defining histogram vectors

$$\mathbf{T^{(u)}} = \left[T_1^{(u)}, ..., T_m^{(u)}\right] \tag{3.1}$$

where every bin is a whole day (midnight to midnight), The sinus is essentially removed, and the gamma pattern remains. The length $m$ of the array would have a significant impact on memory usage and the total number of numerical operations required. By selecting an array length of $m = 10$ days and disregarding all deliveries that claims to have taken longer, the data representation includes most deliveries, while filtering away those of very high lead times due to unforeseen events or poor data quality, that might have caused disturbances in the analysis otherwise.

One thing to note however, is that the weekend "holes" would still be present, and their position dependant on pickup day. There's also an increase in deliveries after every weekend, as parcels unsurprisingly aren't just rarely delivered on weekends but also rarely picked up on weekends. Figure 3.7 below displays the pickup frequency over a full year, where we see that the weekday fluctuations are mostly even greater than monthly fluctuations (Black Friday being an exception).

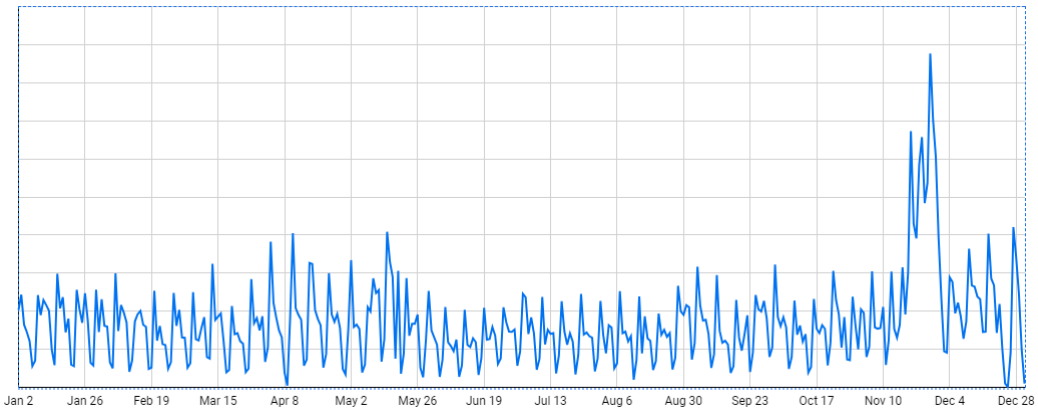



**Figure 3.7:** Parcel pickup frequency over a year.

The phenomenon to be careful about goes under the name *Simpson's paradox*. Simpson's paradox is described by R. A. Kievit et al. as

> ... a counterintuitive feature of aggregated data, which may arise when (causal) inferences are drawn across different explanatory levels: from populations to subgroups, or subgroups to individuals, or from cross-sectional data to intra-individual changes over time. [6]

A simple example is illustrated in figure 3.8 below. The data can be split up in the red and blue subgroup. When analysing the correlation of the whole set, the variable $Y$ appears to grow as $X$ grows. However, when performing the same analysis on each subgroup before merging the data, the correlation appears to be the opposite.

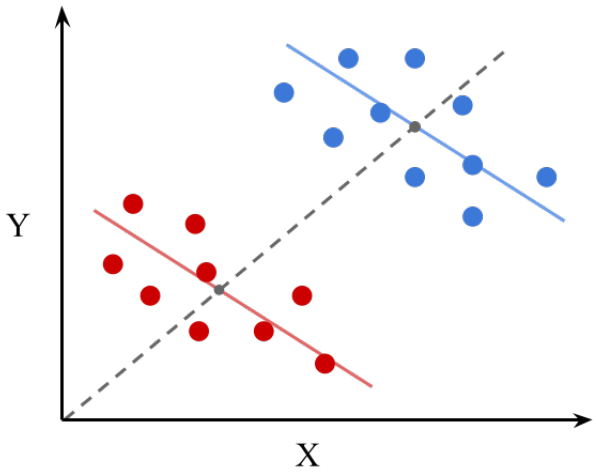



**Figure 3.8:** A simple illustration of Simpson's paradox.

To avoid this paradox over days of the week, the lead time data will remain separated by which weekday the delivery was picked up by the carrier. Any chosen dissimilarity

measure will be executed for between nodes for each weekday separately, and then merged as a final step. This means that the data associated with each node, $\mathbf{T}^{(u)}$, has gone from a $m$ elements long array to a $7 \times m$ matrix.

$$\mathbf{T}^{(u)} = \begin{bmatrix} T_1^{(u,\ sun)} & \dots & T_m^{(u,\ sun)} \\ \vdots & \ddots & \vdots \\ T_1^{(u,\ sat)} & \dots & T_m^{(u,\ sat)} \end{bmatrix} \tag{3.2}$$

Every $\mathbf{T}^{(u)}$ then carries information about how many deliveries to the area associated with the node were picked up on each weekday and took each of 0 to $m-1$ days to deliver. Figure 3.9 displays a colored instance of this matrix.

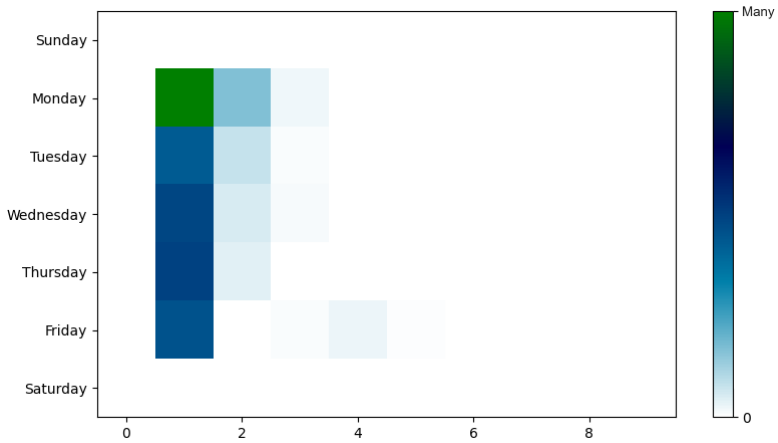



**Figure 3.9:** A figure showing $\mathbf{T}^{(u)}$ for an example instance of $u$.

## 3.2 Custom generated data sets

When determining the options for reliable algorithm to partition sets of positional data into regions, it is helpful to have a way to validate suggested solutions. The tools introduced in section 2.3 provide a way to compare options without supervision, but with algorithm, linkage type, dissimilarity measure, and $k$ all unknown, the complexity easily explodes if one would attempt to try all combinations for each data set. To scale the problem down and assist selection of dissimilarity measure and linkage for REDCAP, combinations may initially be tested on a *simulated dataset* with 5 pre-defined regions to find.

This simulated dataset scatters a group of nodes on a plane, where every node has its own instance of a matrix like the one in figure 3.9. Each histogram vector $\mathbf{T}^{(u,\ day)}$ has $n_{SIM}$ items sampled from a gamma distribution of shape= 10 and scale $= 0.2 + 0.01r$ where $r \in [1, ..., 5]$ depends on which region the node belongs to. To find $r$, H3 was utilized. If sampled node $u$ was found in one out of the 5 pre-defined hexagons within the sample space it would be assigned the $r$-value of that specific

hexagon. If not, its position would randomize again until it is. An example of this generated data with corresponding lead time distributions is displayed in figures 3.10 and 3.11 below.

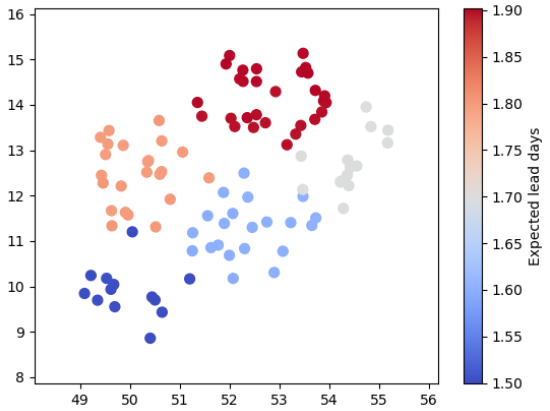



**Figure 3.10:** An instance of generated data with five distinct pre-defined regions.

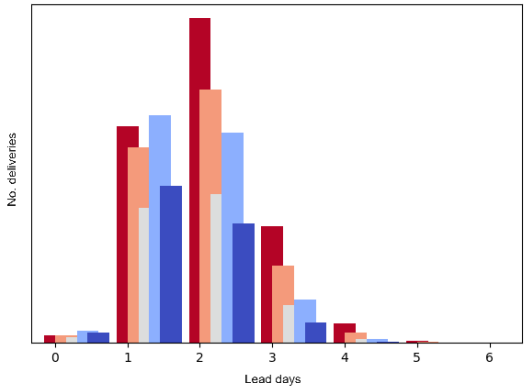



**Figure 3.11:** Lead time distributions split by intended region for the generated data set.

## 3.3 REDCAP implementation

The REDCAP algorithm was implemented as follows:

**Input:**

$S$: The full set of nodes

$k$: The number of requested regions

**1.** Pre-compute $d^{(u,v)}$ according to equation selected dissimilarity metric $\forall u, v \in 1, ..., N$ and store the values in a weight matrix $\mathbf{W}$.

**2.** Determine the adjacency graph $G$ by utilizing `Scipy.spatial.Voronoi` on the coordinates of the nodes in $S$.

**3.** Add all edges in $G$ to a priority queue $Q$, sorted by $d^{(u,v)}$ for edge $e_{uv}$ in ascending order.

**while** *MST not complete* **do**

**4.** Pop the edge $e_{uv}$ with the smallest weight from $Q$. Whenever any of these two conditions are met:

- Nodes $u$ and $v$ already belong to the same connected component
- $W_{uv}$ has received a new value that no longer equals $d^{(u,v)}$

discard the edge and repeat this step.

**5.** Append $e_{uv}$ to the MST and join $U$, the connected component including $u$ with $V$, the connected component including $v$, to a new connected component $C_{new}$.

**6.** Update all weights in $\mathbf{W}$ that connects any existing connected component $C_i$ in the MST to $C_{new}$ according to selected linkage type:

- single-linkage: $\underset{a \in C_i,\ b \in C_{new}}{min} \{d_{ab}\}$
- complete-linkage: $\underset{a \in C_i,\ b \in C_{new}}{max} \{d_{ab}\}$
- average-linkage: $\underset{a \in C_i,\ b \in C_{new}}{mean} \{d_{ab}\}$
- joined-region-linkage: $d_{C_i Cnew}$

**7.** Add all updated edges that are also $\in G$ to $Q$, and repeat the loop.

**end**

**8.** Finally, remove the most recently added $k - 1$ edges from the MST. The resulting connected components represent the $k$ suggested regions.

**Algorithm 3:** The implementation of REDCAP.

The fourth linkage type named joined-region-linkage was added for evaluation because of how the H3 node grouping works. Nodes grouped together have their lead time matrices summed

$$\mathbf{T}^{(U)} = \sum_{u \in U} \mathbf{T}^{(u)} \tag{3.3}$$

where $U$ in this case is the resulting node after the grouping, and $u$ serves as an index for all initial nodes in the specific H3 hexagon. Yet, when following REDCAP, only dissimilarities between single nodes are considered (or an average of them, for average-linkage). The joint-region-linkage option work in a similar manner to the H3 grouping by summing together all deliveries for nodes in the same connected component following equation 3.3, norming the lead times to get relative distributions, and then comparing those distributions using selected dissimilarity measure.

Regarding time complexity, the majority of the computation time is spent in step 1, performing at least $N \times N \times 7 \times m$ numerical operations, depending on selected dissimilarity measure. Step 2 takes generally $O(N\ log\ N)$ time for a 2D-space. Step 3 performs $M$ operations with time complexity approaching $O(log\ M)$. Then, a generous upper bound on step 6 is $O(N^2)$. Step 7 has the same upper bound as step 3, and step 8 is $O(k)$. In total, this gives

$$O(7mN^2 + (N-1)(M\ log\ M + N^2) + k) \tag{3.4}$$

One may terminate the while-loop $k-1$ steps earlier to slightly reduce the time complexity to

$$O(7mN^2 + (N-k)(M\ log\ M + N^2)) \tag{3.5}$$

## 3.4 Ant colony partitioning

In section 2.1.3, we were briefly introduced to how ant colony optimization can be used to solve the traveling salesman problem. Although this problem is a bit different, the basic principle of ACO can be used here as well.

Since this approach works with splitting $S$ rather than merging it into clusters, it would make sense to start looking at a new graph $G'$ derived from $G$ as a graph where edges mark borders between two nodes rather than connections. In section 3.1.2, the `Scipy.spatial.Voronoi` framework was mentioned as a tool to define adjacency between nodes, but it can be used for this purpose as well. In fact, the cell borders of a Voronoi diagram makes up an excellent model of this $G'$.

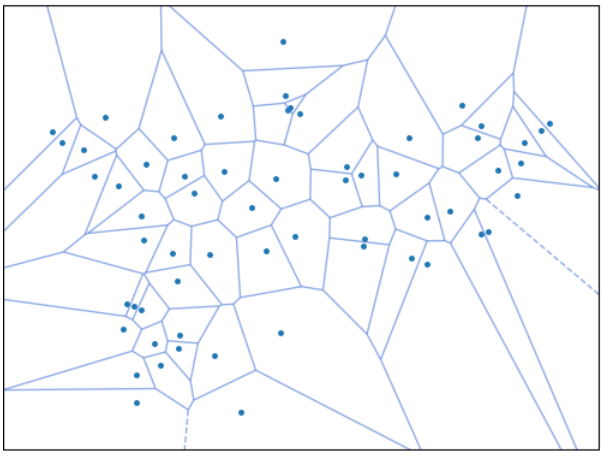


**Figure 3.12:** A second example of a Voronoi diagram.

To avoid confusion between these two graphs, all nodes in $G'$ will be referred to as

*vertices*, and all edges as *ridges*. One may notice that every vertex in $G'$ is the point in the space where the closest nodes are three or more equally close nodes in $G$. Also, every ridge in $G'$ crosses one and only one edge in $G$. This means that some ridges will stretch out towards infinity. `Scipy.spatial.Voronoi` acknowledges this by stating that these ridges connect a vertex in $G'$ with vertex '-1'. The -1 vertex does not exist, but can be treated as if it does. We'll name it the *void* and it's adjacent to all vertices in the outline of $G'$.

**Input:**

$S$: The full set of nodes

$k$: The number of requested regions

**1.** Using `Scipy.spatial.Voronoi` on $S$, extract the ridges to generate the border graph $G'$. Initialize all pheromone levels $\tau_{ij} := \tau_0$ and all $\eta_{ij} := d^{(i,j)}$ following chosen dissimilarity measure.

**while** *Less than K iterations without improvement* **do**

**2.** Initiate $N$ number of ants, all starting in the void.

**for** *all ants* **do**

**while** $k_{enclosed} < k$ **do**

**3.** Select a random unused edge $e_{ij}$ from current vertex $i$ with probability with probability of selecting edge $e_{ij}$

$$p(e_{ij}|S') = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{v_l \notin S'} \tau_{ij}^{\alpha}\eta_{ij}^{\beta}}, \tag{3.6}$$

where $S'$ is the set of visited edges. Add it to the ant's path and update current position to vertex $j$.

**if** *stepped into the void* **then**

**4.** Increment $k_{\text{enclosed}}$ by 1, and add all so far visited nodes to the void. The ant may in the next iteration step out from the void to any vertex adjacent to the updated void.

**end**

**end**

The current path network of the ant now separates $S$ into the region set $\mathbf{R}^{(\text{ant})}$ of size $k$.

**5.** Compute a path score as

$$\mathcal{L}_{\mathbf{R}}^{(ant)} = exp\left(\sum_{r=1}^{k}\sum_{days}^{7}\sum_{u\in R_r} log(P(X_1 = T_1^{(u,\ day)}, ..., X_m = T_m^{(u,\ day)}))\right) \tag{3.7}$$

where $P(...)$ (see equation 2.21 for reference) is the probability that $n_{u,\ day}$ samples are distributed exactly like the histogram vector $\mathbf{T}^{(u,\ day)}$ if drawn from the combined distribution of region $R$

$$\tilde{\mathbf{T}}^{(R,\ day)} = \frac{1}{n_{u,\ day}} \sum_{u\in R} \mathbf{T}^{(u,\ day)}. \tag{3.8}$$

**end**

**6.** Update all pheromone levels

$$\tau_{ij} := (1-\rho)\tau + \sum_{\text{ants}} \mathcal{L}_{\mathbf{R}}^{(\text{ant})} \tag{3.9}$$

**7.** If a new path score record was broken, store away the new score and its suggested region partitioning $\mathbf{R}^{(\text{ant})}$.

**8.** Optional: Slightly increment $\alpha$ for progressively less exploring.

**end**

**Algorithm 4:** The ant colony partitioning algorithm.

Figure 3.13 below displays an example path after completing the inner while loop for one individual ant, $k = 3$, and the data set from figure 3.12. The ant started walking the red path. After stepping into the void (exiting the image), it had closed off the regions A and BC, ticking the variable $k_{\text{enclosed}}$ up from 1 to 2. Since the requested amount of regions in this case is 3, the ant emerged again, this time along the green path. At this step, the red path is included in the void. After its next encounter with the void, the inner while loop will be exited and the ant will receive a score based on the quality of the three regions A, B and C.

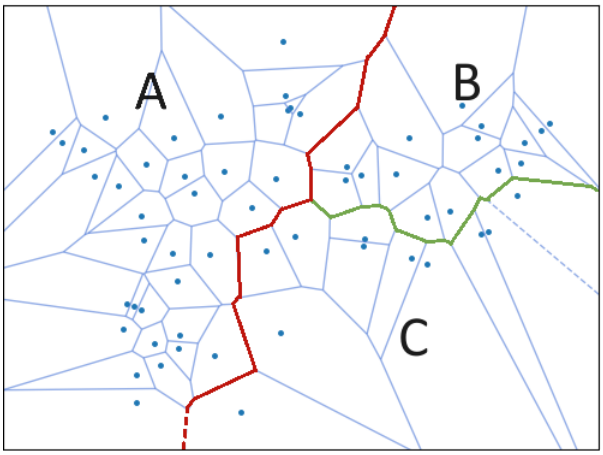



**Figure 3.13:** An example of one ant partitioning of figure 3.12.

Since this is a stochastic algorithm, it the time it takes to converge varies for different runs, and each iteration varies in length depending on the selected route of the ant. It is therefore hard to motivate a solid time bound for the algorithm. Testing the algorithm has shown that it at least takes generally longer to converge to a result than it does for REDCAP, which is to be expected.

The stochasticity is also an obstacle when verifying a solution. Due to differences in the outcome of every run with no change in initial conditions, consensus clustering isn't a feasible approach to find the optimal $k$. In section 4.4 we'll see how the ant partitioning algorithm can outperform REDCAP in total likelihood $\mathcal{L}_{\mathbf{R}}$, but that the regions suggested by REDCAP are still preferable. This, together with the high computation time of the ant colony algorithm has led to this study only exploring the ant colony approach for a given fixed value of $k$. This motivates the choice of using the likelihood as a path score in step 5. When the number of regions doesn't vary, a penalty for $k$ becomes irrelevant, and simply $\mathcal{L}_{\mathbf{R}}^{(\text{ant})}$ suffices as an objective quantifier.

## 3.5 Testing procedure

So far, we've listed a few possible approaches to the regionalization problems, each with their own range of options. Since other studies have already proven REDCAP to perform better and faster than SKATER, there will be no need to evaluate

SKATER as an option. ACO may offer good results, but with the long computation time and inconsistent outcome, it is less appropriate for comparing dissimilarity measures for example. Instead, these parameters, including its own: constraint and linkage types, will be tested on a couple of data sets listed in table 3.1 using REDCAP, and evaluated using Consensus Clustering.

In total, there are four different dimensions of options to evaluate: dissimilarity measure, constraint, linkage and $k$. The result will be evaluated using the relative PAC increase $\mathcal{I}_k$ (see section 2.3.2). Since combinations of parameters that wields the $\mathcal{I}_k$ may vary depending on the dataset used, a brute force iteration of all combinations will be used. However, some options may consistently fail to yield a decent outcome for any combinations of the other parameters. In those cases, these options will disqualify as candidates, and not be further evaluated as the exploration proceeds with new data sets. After the tests, a successful result is achieved if majority voting (see section 2.3.2.1) between the remaining candidate setting combinations yield partitionings that are reasonable according to the specifications from section 1.2. Additionally, the number of remaining combinations should be kept as low as convenient, to avoid unnecessary computation.

The data sets that will be explored are listed below. The first set is a generated set (see section 3.2) with five easily distinguishable regions. The second set has similar structure, but with much lower value of $n$ for noisier distributions and more unclear regions. The third and fourth are historical data with deliveries to Sweden, with different number of nodes filtered away. The fifth is also Sweden, but deliveries by all carriers except one are filtered away. The sixth data set collects data from a number of carriers in Germany, where nodes again have 500+ deliveries each.

Data set 1-4 will be used for determining viable setting combinations of REDCAP. As a measurement of performance, consensus clustering will be used, with $H = 100$ consensus iterations and $(x_1, x_2) = (0, 1, 0.9)$ as PAC interval. The PAC-value will be plotted over a range of $k$-values for all combinations of linkages, constraints and dissimilarity measures, with the hope of revealing the true amount of regions $k_{\text{true}}$ as the highest $k$ before PAC shoots up.

Data set 5 will be used for comparison with the carrier's estimated (or self-suggested maximum) lead time, and data set 6 will be used to evaluate an instance of ant colony partitioning's performance compared to REDCAP.

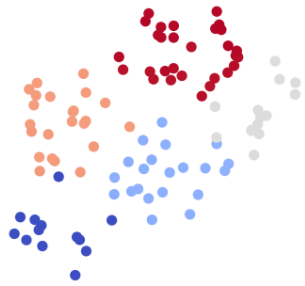
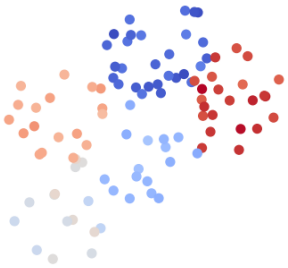
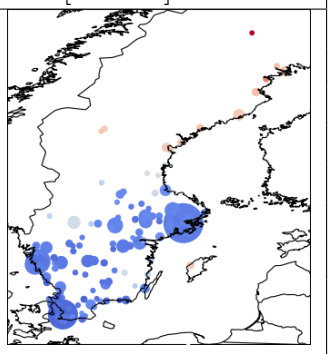
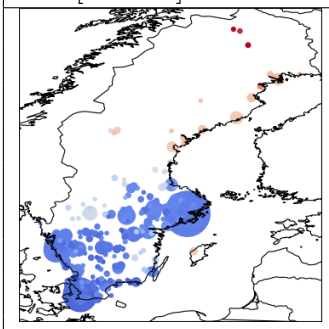
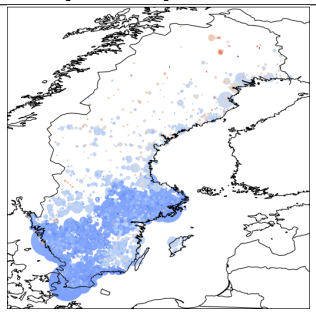
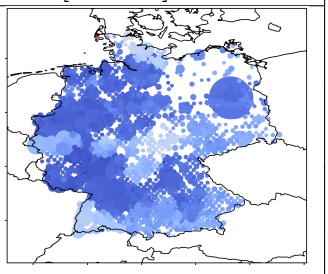
| Distinct example data | Confused example data | Sweden 500+ |
|---|---|---|
| N 100<br>n $2.1 \cdot 10^7$ | N 100<br>n $2.1 \cdot 10^5$ | N 144<br>n [hidden] |
|  |  |  |
| **Sweden 250+** | **Sweden full resolution** | **Germany H5** |
| N 246<br>n [hidden] | N 7418<br>n [hidden] | N 1224<br>n [hidden] |
|  |  |  |

**Table 3.1:** A table of data sets used in evaluation of partitioning strategies.

# 4

# Results

## 4.1 REDCAP option elimination

Figure 4.1 below displays the PAC-result for data set 1 (table 3.1), the distinct example data.

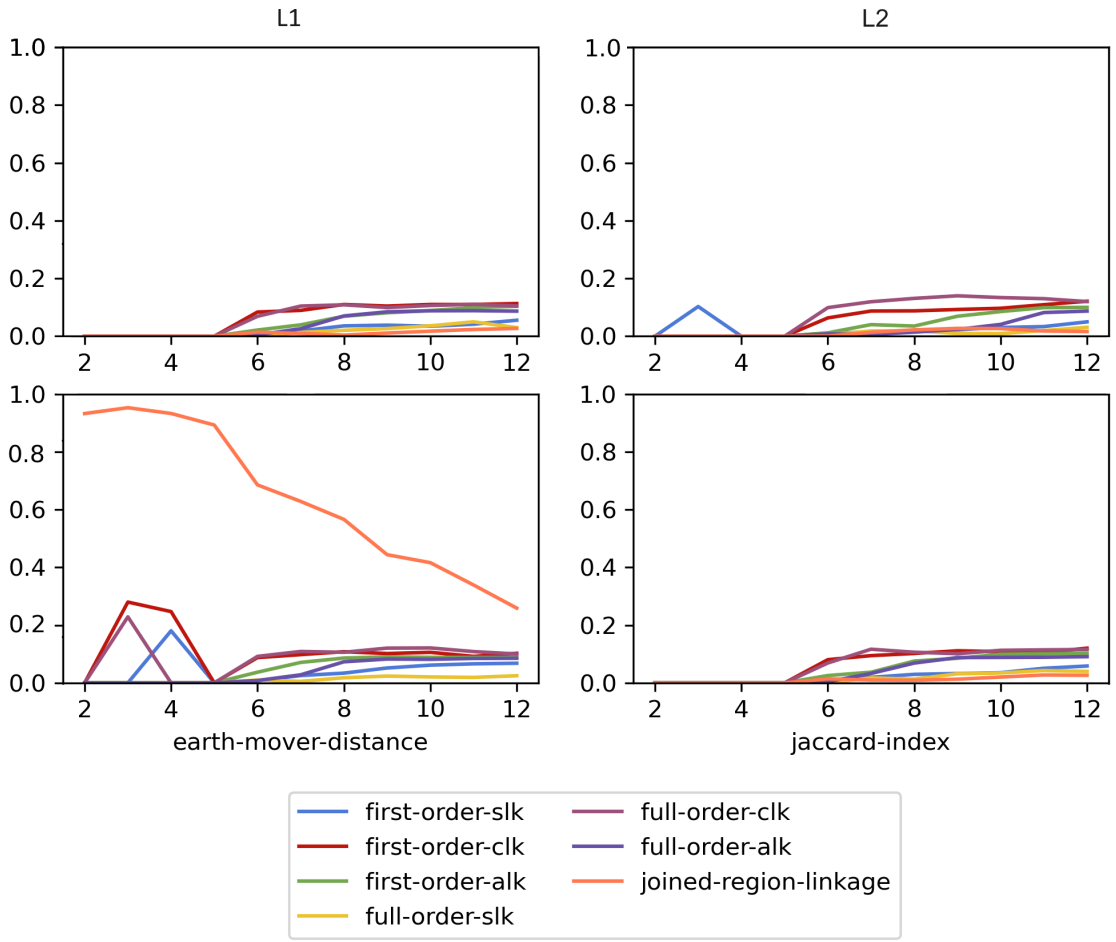



**Figure 4.1:** PAC over $k$ for a range of options for data set 1.

Some observations that arise are listed below.

1. The PAC-values are generally low for most curves.
2. Earth mover distance joined region linkage is an exception.
3. Most PAC-values are exactly 0.0 until $k = 5$.
4. There are no PAC-values at $k < 5$ that are very close to, but larger than 0.
5. PAC-values for single-linkage and joined-region-linkage (except when paired up with EMD) maintain very low values even after $k = 5$.

1. and 3. can be explained by the large value of $n$ making it easy to distinguish the distributions of each node, causing little confusion around which node belongs to which region. 2. is an effect of the fact that for joined-region-linkage

$$n_U = \sum_{u \in U} n_u \tag{4.1}$$

for any connected component $U$ in the procedure of building the MST (see section 3.3), and EMD is the only dissimilarity measure that doesn't work with normalized distributions. The reason behind 4. was mentioned in section 2.3.2, being that for a dataset of regions this easily distinguishable, the only confusion for $k < k_{\text{true}}$ is which of the complete regions to group together. 5. tells us that single-linkage clustering is still consistent with its regionalization even after $k_{\text{true}}$ is found, suggesting that if we aim to use single-linkage, then PAC cannot reliably help finding the best $k$.

Now we'll see the same result plots for data set number 2., the Confused example data, where the total amount of the deliveries is just a hundredth of those in data set 1.

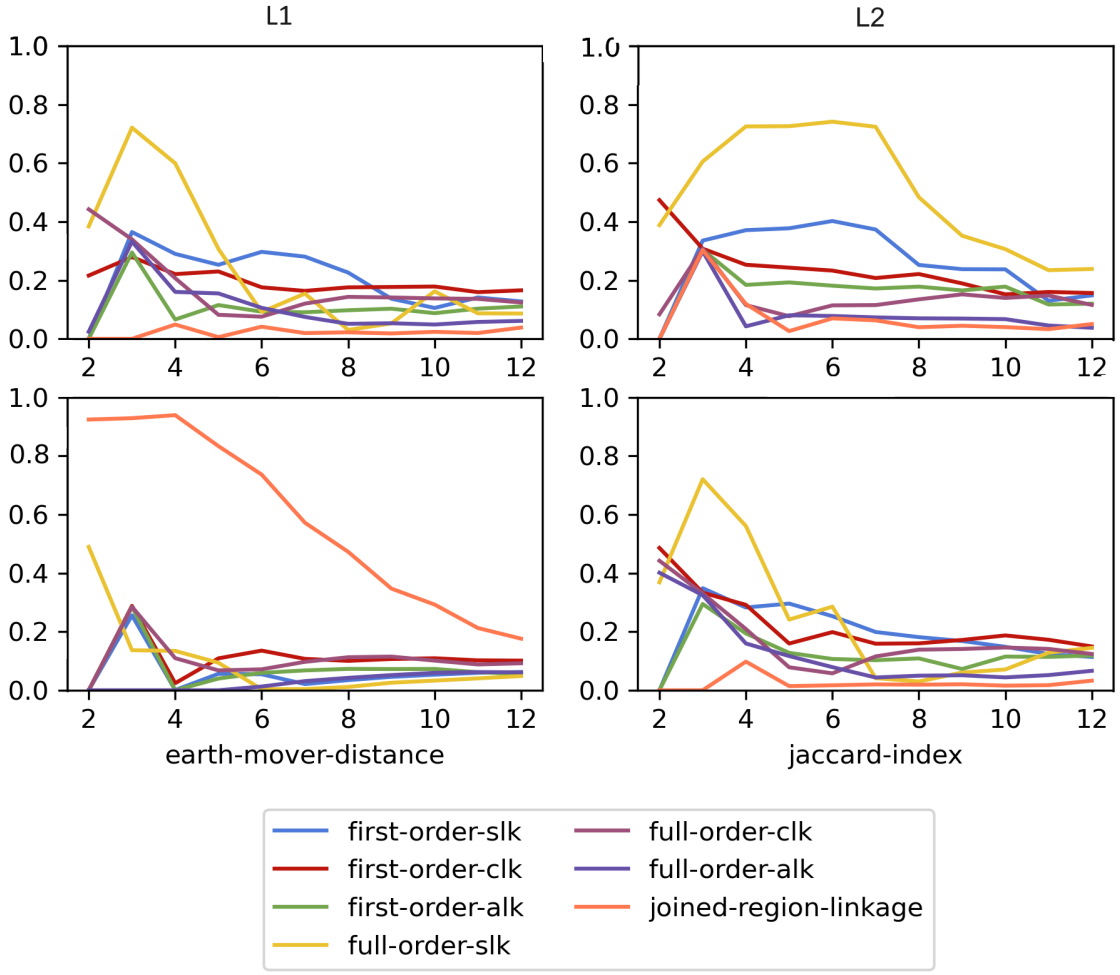



**Figure 4.2:** PAC over $k$ for a range of options for data set 2.

Overall the values are much higher than before, but more important is the fact that the different plots do not seem to agree on the optimal $k$. For EMD, most linkage types seem to suggest $k = 4$ is good, but almost all of them seem mostly biased towards $k = 2$. This also applies to some extent to a couple of more options. Not many combinations consider 5 to be the optimal $k$ and even fewer have perfect consensus for $k = 5$. All confusion matrices for $k = 5$ can be found in appendix B, where one can confirm that only EMD full-order average-linkage maintains perfect consensus over the five intended regions.

A takeaway from the two tests displayed in figure 4.1 and 4.2 is that single-linkage and EMD joined-region-linkage both performed poorly. They're both very confused overall and get gradually less confused with increased k, rather than more confused as we saw in the ideal behavior in figure 4.1. Single linkage is hereby disqualified. Since joined-region-linkage does enough computations to execute much more slowly than the others, and had the lowest increase at $k_{\text{true}}$ on data set 1, we'll disqualify this one as well before moving on.

The next two (and the third) data sets are based on Sweden. Sweden is a great example of a real data set with clear differences in lead times at different locations in the country. The difference in number of nodes between the apparent regions also lets it fool Monti et al.'s method of finding $k_{\text{true}}$ using the difference in area under the CDF graph (see appendix A). Figure 4.3 below displays the PAC evolution for data set number 3.

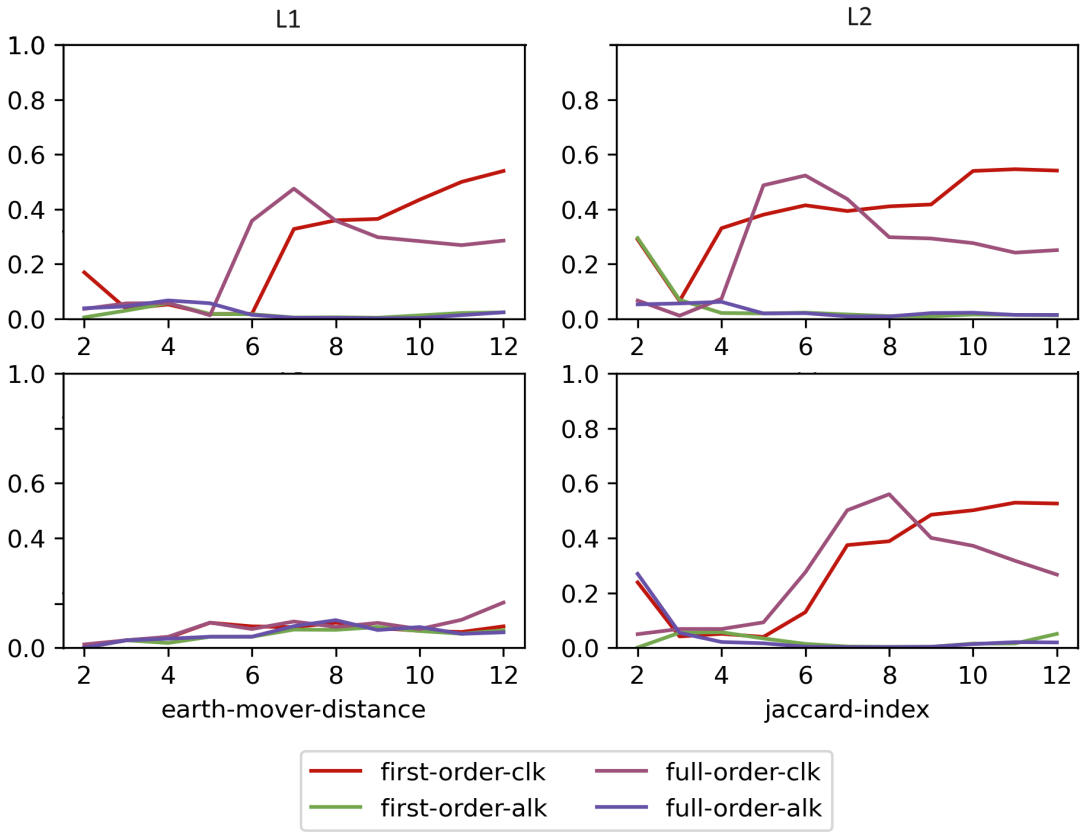



**Figure 4.3:** PAC over $k$ for a range of options for data set 3.

Here, we see that neither linkage types nor dissimilarity measures seem to agree on which $k$ to select. $L_2$ first-order and full-order complete-linkage suggests $k = 3$ and $k = 4$ respectively, three combinations suggests $k = 5$, and $L_1$ first-order complete-linkage suggests $k = 6$. Figure 4.4 below displays each of these partitionings, with non-diverging parts of Voronoi cells colored by region. The region colors are selected from a set of distinct colors to help illustrate which region each node belongs to, and doesn't represent any color scale value.

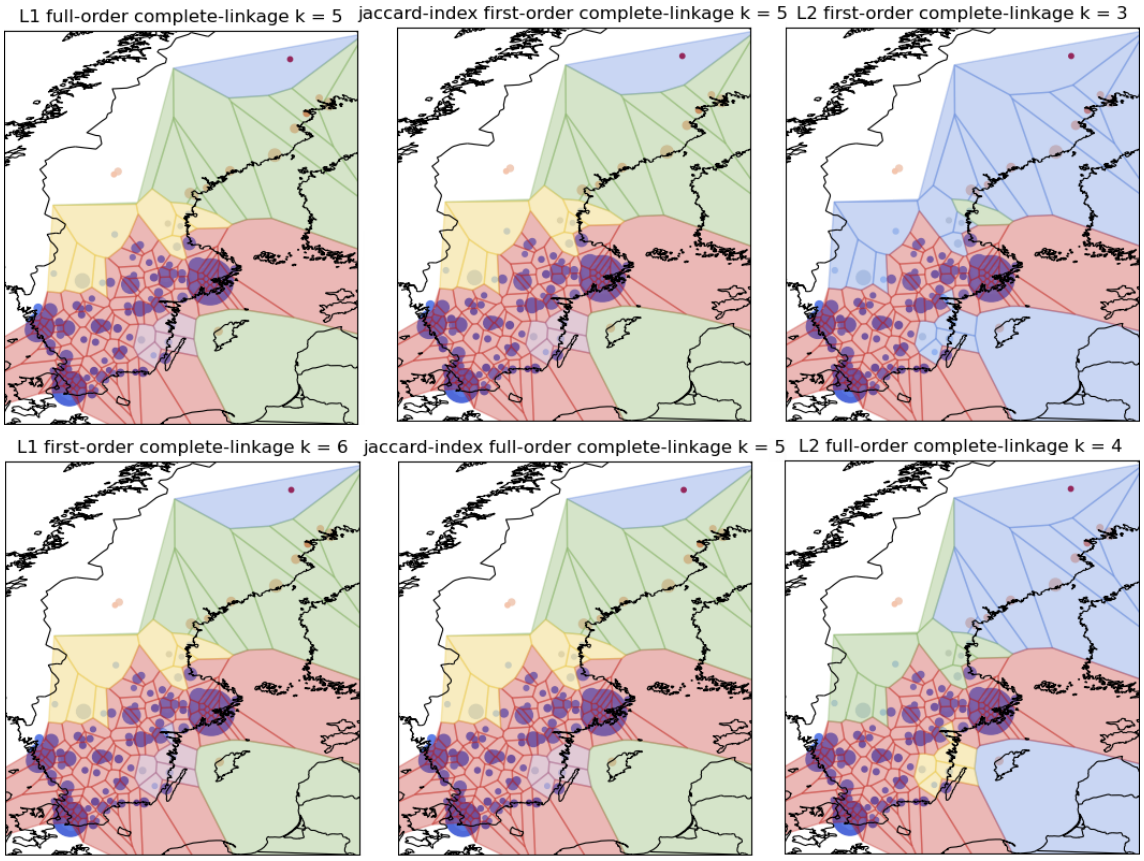



**Figure 4.4:** Six partitionings of data set 3 with different options.

The top right partitioning is the only one that suggested $k = 3$. There, it has separated locations with low and high average lead time into two main regions. Then one single node makes up the third region. When looking at the other partitionings, we see that four of them have isolated a single node as well. However, this node has a warmer color than its neighbors, and all these four partitionings agreed to isolate it. In other words, for this set, $L_2$ first-order complete-linkage didn't do very well. The bottom left partitioning isolated an extra single node compared to the rest. Bottom right included the north most red node in a larger region. All option combinations that suggested $k = 5$ found the same partitioning.

By applying majority voting (see section 2.3.2.1) giving each of these six combinations one vote each, we get the result in figures 4.5 and 4.6 below.

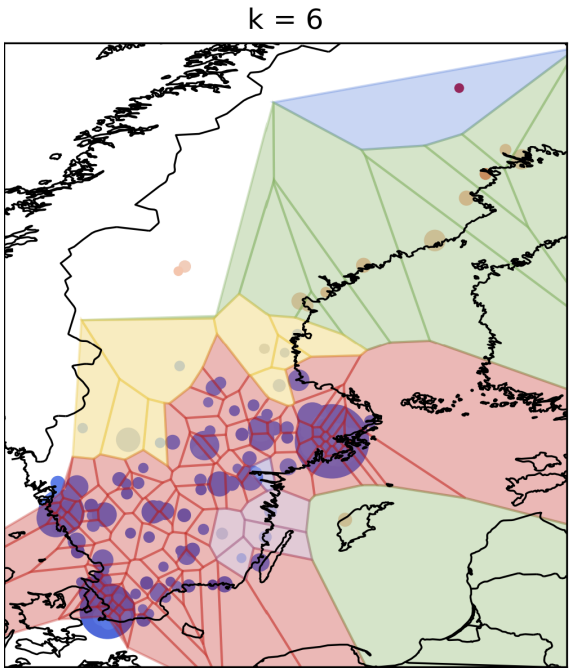



**Figure 4.5:** Partitioning after merging suggestions from figure 4.4 by majority voting.

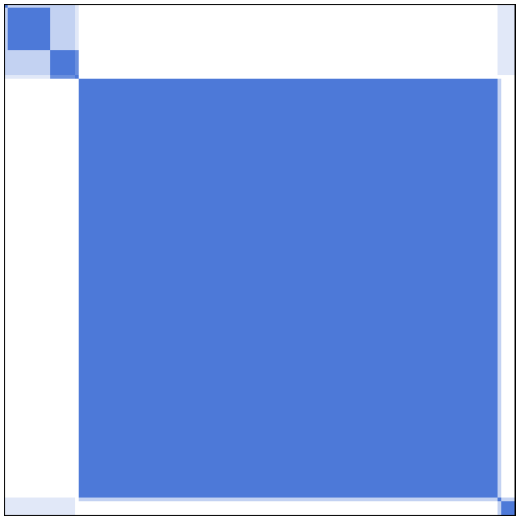


**Figure 4.6:** Co-association matrix from the six suggestions in figure 4.4.

This result reveals one flaw with the majority voting approach. One single node north of the purple region has a slightly more blue color than its neighbor to the south. This node was assigned a region of its own, despite the fact that it wasn't the case in any of the suggestions. This node is represented as the single blue pixel between the large square and the bottom right square in the co-association matrix. What we see is that this node shares equal belongings to both clusters, and therefore didn't win a majority in either. As a result, it was left alone. This can be easily bypassed by a simple addition to the implementation of the algorithm: Given co-association matrix $C$, any node $u$ that has 50 % belonging to some other node(s) $V : v \in V \Leftrightarrow C_{uv} = 0.5$ will be paired up with the region that contains the node in $V$ most similar to $u$.

Further, we'll see how partitioning and $k$ is also sensitive to the input data. Figures 4.7 and 4.8 below shows the same plots for data set 4, which is identical to data set 3, except some additional nodes, all with $n_u$ lower than those in data set 3.

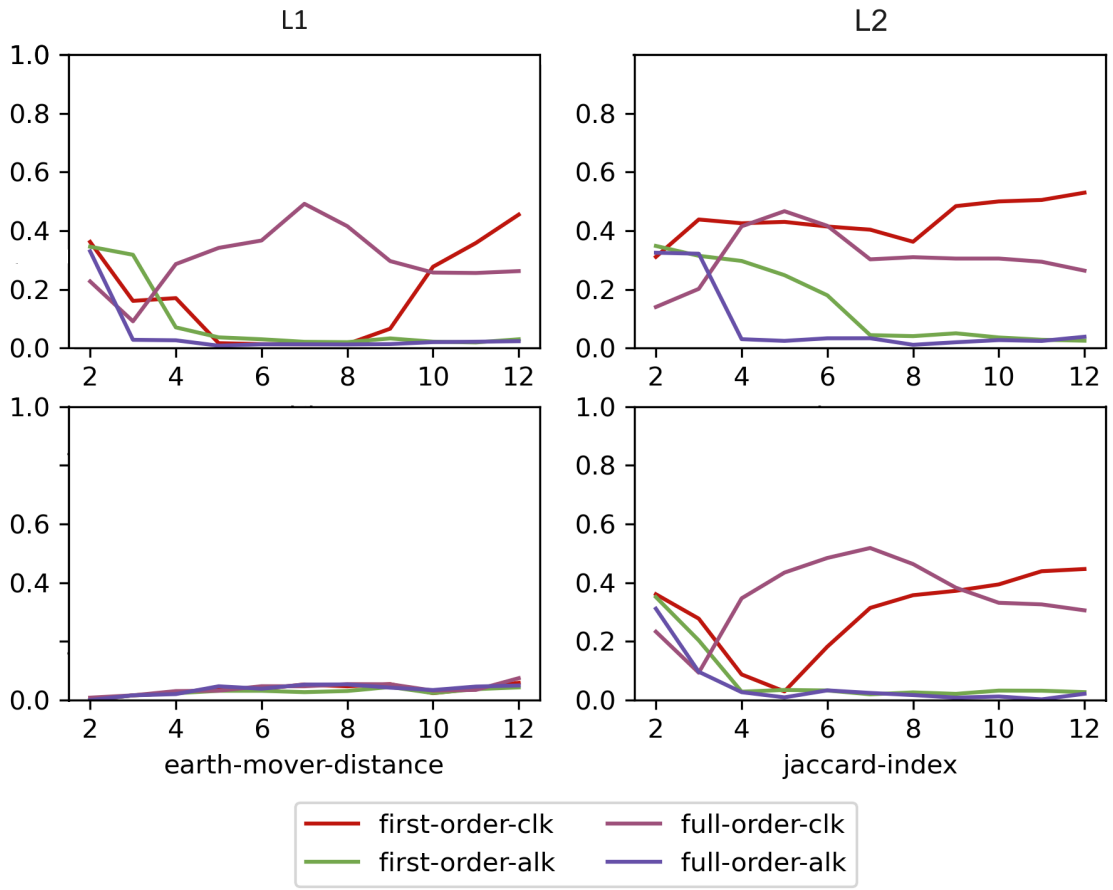



**Figure 4.7:** PAC over $k$ for a range of options for the data set 4.

Here, the same options do not at all stick to their suggestions of $k$ from data set 3. The most voted $k$ is now $k = 3$, but the three options that suggested it do not even fully agree on which three regions to select. The result of majority voting for these six options is displayed in figure 4.10, with $k = 4$. There is again one single node that has a region by itself (marked as node $a$, south-east of the green region), but this one seems more similar to its neighbors than the red node in figure 4.4. In fact, in figure 4.8, we see that for five of the six suggestions, node $a$ got partitioned with either the red region or the green region (displayed in different colors in figure 4.8), but top left partitioning put it alone, and in total no majority wanted to include it.

Allowing the $L_1$ first-order complete-linkage combination to vote for this set can seem a bit strange, since it decided to leave as much as 4 seemingly insignificant nodes by itself. The figure set in appendix C collects all partitionings for $L_1$, $L_2$, jaccard-index, first-order constraint, full-order constraint, complete-linkage, and average-linkage for data set 4 (12 combinations total). These figures reveal that the tendency of leaving single nodes by themselves is significantly more common in options using either average-linkage or first-order constraint. Figure 4.9 displays the result obtained by revoking the votes of these options (leaving three votes in total), being the same as in figure 4.10, but $a$ is now included in the green region. Figure 4.11 displays the

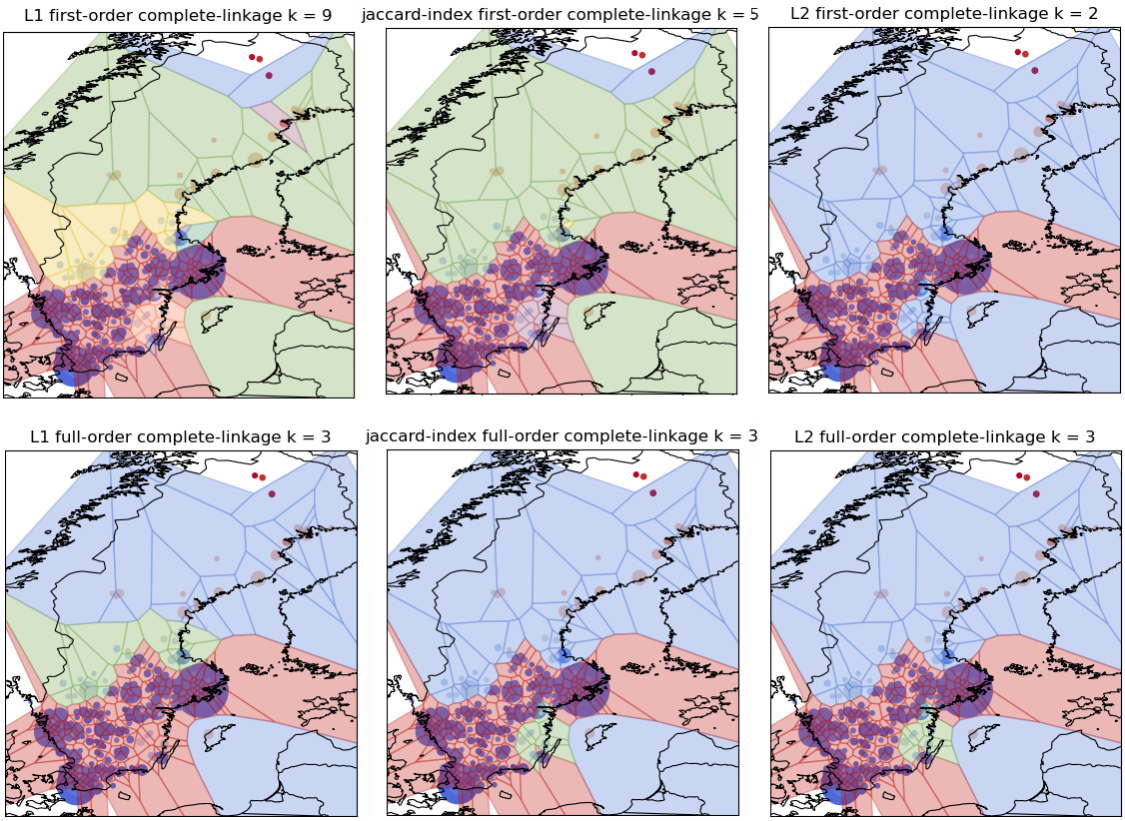



**Figure 4.8:** Six partitionings of data set 4 with different options.

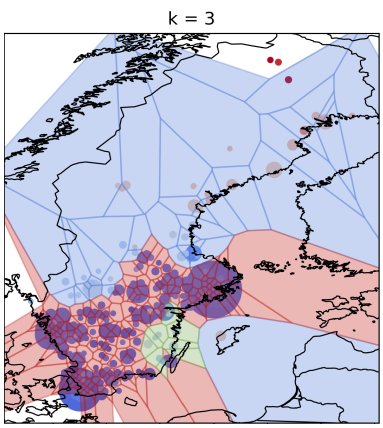



**Figure 4.9:** Data set 4 partitioned after 3 votes.

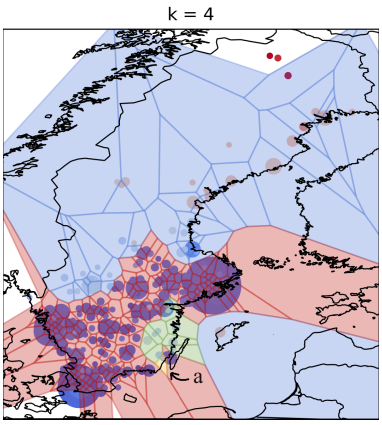



**Figure 4.10:** Data set 4 partitioned after 6 votes.

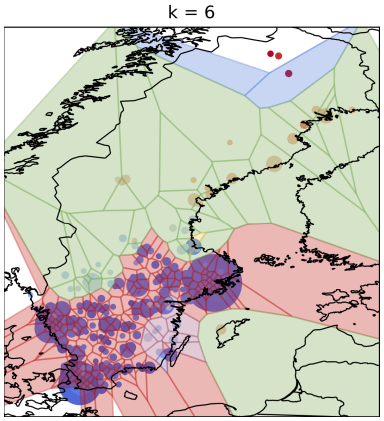



**Figure 4.11:** Data set 4 partitioned after 12 votes.

result of letting all 12 combinations in appendix C vote.

## 4.2 Comparison with carrier specified lead times

Figures 4.12 - 4.14 display how well the time estimations specified for each zip-code location matched the actual lead times. Figure 4.12 displays deliveries from a single carrier throughout Sweden. Figure 4.13 displays the estimations of lead time days given by the carrier (averaged over weekdays) in the same color scope, and figure 4.14 displays the result of subtracting the estimations from the actual lead times to display how well the deliveries were on time. The gray dots imply that the deliveries were on average delivered on the same day as specified, blue color means they're generally delivered earlier and orange color means they tend to be late.

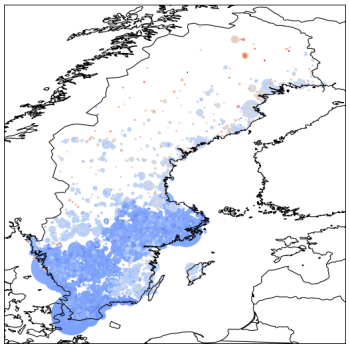


**Figure 4.12:** Data set 5 with no grouping or filtration.

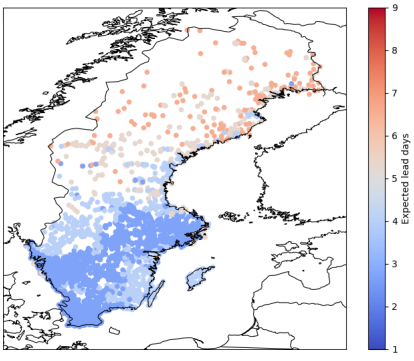



**Figure 4.13:** Carrier specifies estimations for data set 5.

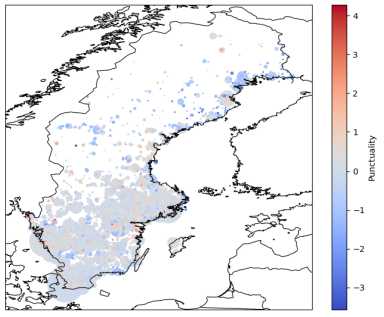



**Figure 4.14:** Difference, representing how well deliveries in figure 4.12 were on time.

What we see is that the estimations are mostly quite accurate. The bluer area in southern Sweden with shortest estimations seems to align perfectly with the

historical data. Only in the north with the orange 6 days estimations does the carrier have a tendency to deliver the package earlier than estimated.

We may also compare the estimations (figure 4.13) to the partitioning results (mainly figure 4.5 and 4.9) where we see that the variance in estimations does align with the found regions quite alright. Most of the confusion happens in the north. The part north of the red cell region common to figure 4.5 and 4.9 seems to be split into two regions with a vague border in between in the estimations, one region in figure 4.9 and three regions in figure 4.5. This is likely an effect of the sparse data record in this area. However, another observation is that the estimations seem to imply the same lead time for Gotland (eastmost large island) and most of Småland (mainland area closest to Gotland) while nearly all partitioning suggestions so far have assigned them to separate regions.

## 4.3 Performance on other data sets

Figures 4.15 to 4.18 below showcase the result of running the algorithm that partitioned data set 4 into regions displayed in figure 4.9. The region colors are here omitted, to avoid obstructing the node colors. Instead, region borders are drawn using thicker black lines.

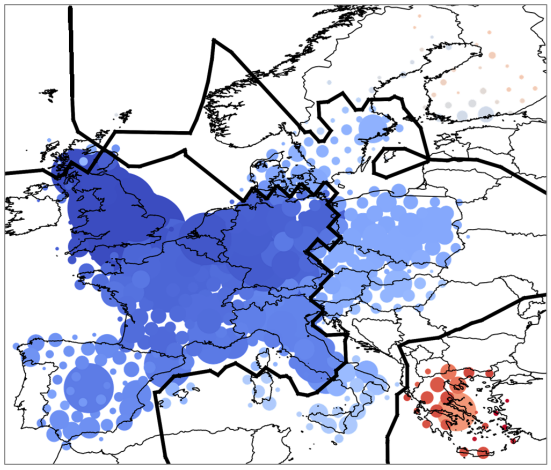


**Figure 4.15:** Partitioning of Europe.

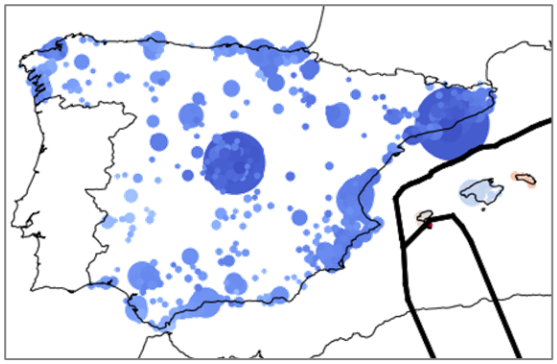


**Figure 4.16:** Partitioning of Spain.

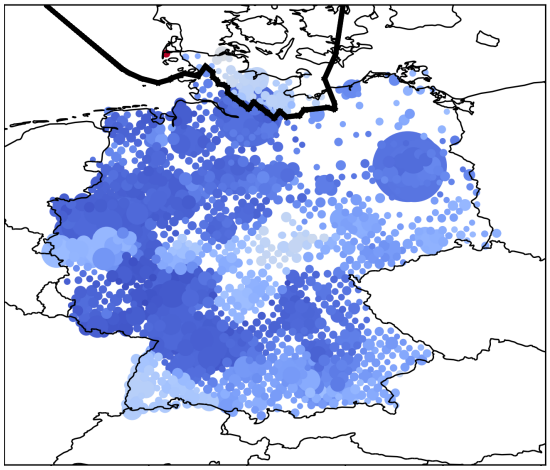


**Figure 4.17:** Partitioning of Germany

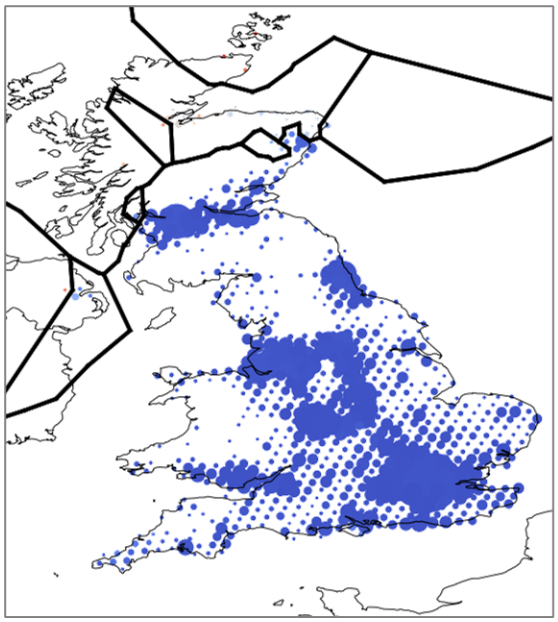


**Figure 4.18:** Partitioning of UK.

The algorithm does well in capturing the most significant lead time differences. For the case of Spain, UK, and Germany, these differences are mostly between islands and the mainland (or remote locations in the north for UK).

## 4.4 ACO results

The ant colony approach provided a powerful tool to optimize a partitioning of a data set into $k$ regions with respect to some objective function. However, at that point one may question the objective function as a quantifier. Figures 4.19 and 4.20 below displays the result of letting ant colony partitioning and REDCAP $L_1$ full-order-clk divide data set 6 (Germany) into $k = 8$ regions. The likelihood score (equation 3.7) for REDCAP was $7.8 \cdot 10^{-21}$ and ACO scored $3.6 \cdot 10^{-18}$

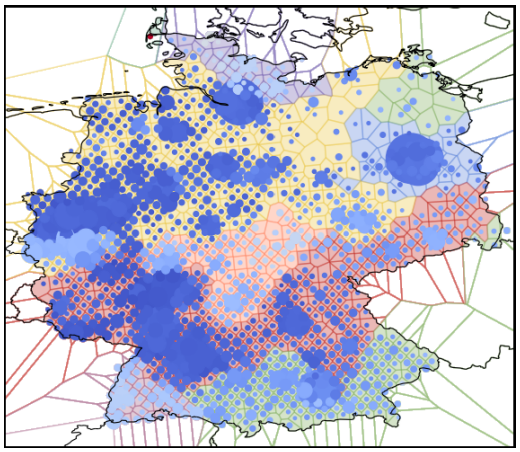


**Figure 4.19:** Germany partitioned into 8 regions by REDCAP.

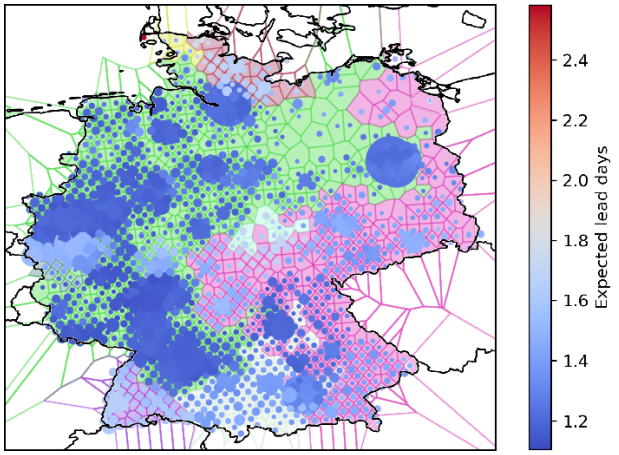



**Figure 4.20:** Germany partitioned into 8 regions by ants.

Data set 6 of Germany expressed some apparent, but more subtle spatial differences in lead times than Sweden did, making it an excellent set for this comparison. What we see is that the most apparent differences, such as at the north most and south west most parts of Germany, are caught by both algorithms. However, at Germany's eastern side, it is slightly more difficult to tell where the region borders should be drawn, and the two algorithms are at a disagreement. Intuitively, one would say that REDCAP has done a better job, as its region shapes look more soothing and similar to how county regions in general look like. The ant colony partitioning's borders are more complex, and twists back and forth around many of the nodes, although this gave it a much higher likelihood score.

The effect we see resembles a case of what, in the topic of machine learning, is called *overfitting*. Overfitting happens when machine learning models aim to generalize some concept (classification, regression, etc.) based on some set of existing data, but it follows that data too precisely, and therefore doesn't generalize very well. If one would add a couple of new deliveries at the south western part of Germany and ask the algorithm whether they belong to the white or the pink region, the answer would appear to be nearly random.

# 5

# Discussion

This study has explored the task of finding a reliable algorithm that partitions any given set of spatially distributed lead time data into regions where every region maintain an as similar lead time distribution as possible. The algorithm selected to execute the task goes by the name REDCAP. Although the algorithm proved fast and efficient, it had to be tested on a variety of different parameters, where each test included re-running the algorithm 100 times (although dissimilarity values could be re-used), which slowed down the computation process. Şenbabaoğlu, et. al.'s method of observing the proportion of ambiguous clustering helped identifying suitable $k$-values, although historical data over Sweden proved that the seemingly optimal $k$ depends on dissimilarity measure, constraint and even some filtering adjustments of the data. Why this is the case is yet to be answered. Maybe it is an effect of the randomized adjustments to the set in the consensus clustering procedure. Maybe it's not. There are many steps where results may part ways on the journey from a complex problem to a simple answer.

To instead apply the AIC or BIC as a tool to quantify solutions and find the optimal $k$ would both be much faster consensus clustering and fully deterministic. It was, however, unfortunately discarded as an option due to the usual application of the criteria being too distant from this one. When defining a model for a set of data, there's only the data and the parameters to take into consideration. However, in this case, we have both the number of deliveries $n$ and the number of nodes $N$ unrelated to each other that both have a negative effect on the log-likelihood $log\ \mathcal{L}_{\mathbf{R}}$. Some empirical experiments suggested that $n$ affects $log\ \mathcal{L}_{\mathbf{R}}$ linearly and $N$ logarithmically. Adding a factor $n\ log\ N$ to the penalty could then balance out this effect, but $k(n\ log\ N)$ alone yields a too large value, to the degree that for no tested instance of $S$ the optimal suggested $k$ was ever different from $k = 1$. Scaling the penalty term would have been an option, but determining a scaling factor without applying any bias towards the selection of $k$ is scientifically difficult.

The theme of consensus was instead continued upon, as a tool to unite the different partitioning suggestions that had occurred. To identify one single solution, majority voting was applied among some candidate option combinations that had a tendency to perform better than others. Whenever $k$ is tested for values larger than $k_{\text{true}}$, all combinations tend to cut away single nodes from the major, and assumably

*true* regions. The option combinations were entrusted a vote or not depending on whether they tended to involve single cut off nodes, that others didn't agree on, or not. The number of passed votes would also preferably stay low to avoid unnecessary computing. In the end, only full-order complete-linkage paired up with the three dissimilarity measures $L_1$, $L_2$ and jaccard-index remained, making three votes in total. By maintaining an odd number of votes, we also help out avoiding the phenomenon revealed in figure 4.6 where a single node gets voted equally towards two separate regions and ends up alone. This is however still possible if the three suggestions have one node belonging to three separate regions, one for each suggestion. This behavior is something we were warned about by Gao et. al., as a risk of using majority voting for merging clusters. One possible alternative to the majority voting approach would be to take the co-association matrix in figure 4.6 and interpret it as a weighted adjacency matrix of a so called *hypergraph*, and then apply a clustering algorithm on this hypergraph, for example k-means, on this hypergraph to determine the final clusters.[3] Anyone who wishes to further develop this regionalization algorithm is strongly invited to explore such possibilities.

As an option to the REDCAP algorithm, an adaptation of Ant Colony Optimization was lightly tested as a candidate solution to the problem. The strength of this algorithm was its ability to continually tweak borders and search for a better solution. The quantifier used that measures goodness level of a solution looked at the probability that the lead time distributions of every node within a suggested region belonged to the combined distributions for the region in question. The downfall of the algorithm was that optimizing this likelihood wasn't of priority, combined with its high computation time. Another reason to its downfall was that this study failed to acquire an objective function that computes a fair partitioning score for any value of $k$, making the current implementation of the algorithm incomplete for the task. Consensus Clustering couldn't be utilized due to too much stochasticity in the results, combined with exploding large computation time. AIC and BIC proved too distant to the problem in question, and wouldn't penalize large values of $k$ in a reasonable way.

The result yielded from the ant partitioning algorithm does however serve as a proof of concept. It might even be the preferable option to use for some similar problem, where optimizing likelihood or other specified objective function is top priority. Even for this application, some adjustments or future work could make it more viable:

1. By utilizing a programming language that allows for parallelization, the exploration of one ant could run on one computer core at the time, greatly speeding up the process.

2. Approaches could be taken to avoid the overfitting. For example:

   (a) Involve some factor in the objective function that increases for simpler region borders.

(b) Apply a pre-processing algorithm that groups together the most similar nodes.

(c) Test different ants on different data and then merge the results.

Additionally, to make it possible to involve multiple $k$ in the evaluation, one could apply some theory to calibrate an adaptation of AIC / BIC to give a fair penalty to $k$. Then, while exploring optimal regions different ants could be tasked to explore different numbers of regions as well. Then, $k$ could spread differently to the next generation of ants depending on the success of each $k$-value, similar to the principle of genetic algorithms. One would however need to bear in mind that this would add a whole dimension to the problem, and some means (such as greatly increasing the number of ants) would need to be taken to maintain exploration and avoid local minima.

As a conclusion, an algorithm that fulfills the aim of the project has been found. It partitions maps based on historical data, and focuses on the most distinct differences within the given data set. There is however still some sensitivity to the number of nodes in the data, which suggests that there might still be more robust approaches to consider. The number of repetitions in the consensus clustering method of determining number of regions also suggests that there are faster options to achieve partitioning. For future improvements of the algorithm, a recommendation is to further explore objective functions, and see if consensus can be achieved by fewer iterations, possibly leaving a larger amount of uncertain partitioning suggestions, that may be merged in a more advanced way than by majority voting.

# Bibliography


[1] Joseph K. Blitzstein and Jessica Hwang. *Introduction to Probability.* CRC Press, Boca Raton, FL, 1st edition, 2015.

[2] Uber Engineering. H3: Uber's Hexagonal Hierarchical Spatial Index, 2008-07-27.

[3] Caroline X Gao, Shengqi Wang, Ye Zhu, Myriam Ziou, Shu M Teo, Catherine L Smith, Derek Chiu, Aline Talhouk, Sue M Cotton, and Dominic Dwyer. Ensemble clustering: A practical tutorial, Feb 2024.

[4] D. Guo. Regionalization with dynamically constrained agglomerative clustering and partitioning (redcap). *International Journal of Geographical Information Science*, 22(7):801–823, 2008.

[5] Built In. What Is a Voronoi Diagram? A Data Science Explainer, 2023-02-21.

[6] Rogier Kievit, Willem E. Frankenhuis, Lourens Waldorp, and Denny Borsboom. Simpson's paradox in psychological science: a practical guide. *Frontiers in Psychology*, 4, 2013.

[7] Sven Kosub. A note on the triangle inequality for the jaccard distance. *Pattern Recognition Letters*, 120:36–38, 2019.

[8] Wahde M. *Biologically Inspired Optimization Methods, An Introduction.* WIT Press, 2008.

[9] S. Monti, P. Tamayo, J. Mesirov, and T. Dolub. *Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data.* Kluwer Academic Publishers, 2003.

[10] Numpy and Scipy Documentation. scipy.sparse.csgraph.minimum_spanning_tree, Retrieved 2024-02-13.

[11] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. In *Proceedings of the Fourth ACM International Conference*

*on Multimedia (ACM Multimedia)*, page 2. ACM Press, 1996.

[12] G. Câmara R. M. AssunÇão, M. C. Neves and C. Da Costa Freitas. Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees. *International Journal of Geographical Information Science*, 20(7):797–811, 2006.

[13] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66, 1998.

[14] Ernst Wit, Edwin van den Heuvel, and Jan-Willem Romeijn. 'all models are wrong...': an introduction to model uncertainty. *Statistica Neerlandica*, 66(3):217–236, 2012.

[15] Yasin Ṣenbabaoğlu, George Michailidis, and Jian Z. Li. Critical limitations of consensus clustering in class discovery. *Scientific Reports*, 4, 2014.

# A

# Sweden fooling S. Monti et al.'s method of determining $k$

S. Monti et. al. suggested in their paper *Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data* (2003) that one may utilize consensus clustering to find the number of regions $k$ by observing the increase in area under the CDF-curve. However, the CDF curves for data set 3 (see table 3.1) proves this method unreliable.

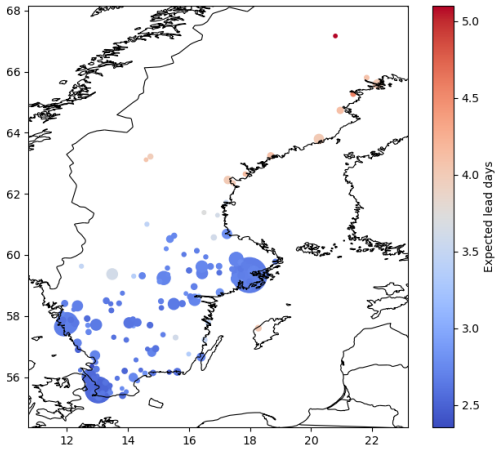



**Figure A.1:** Delivery data for Sweden

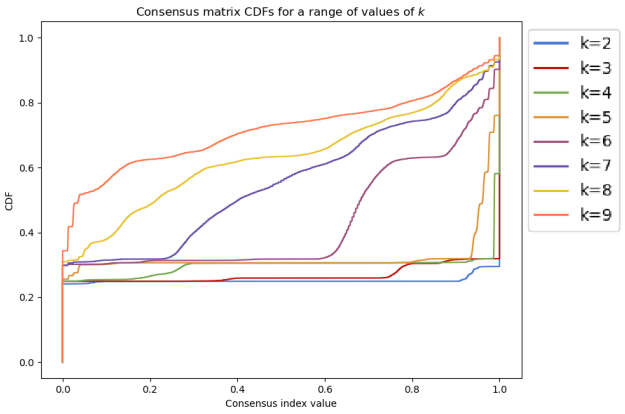
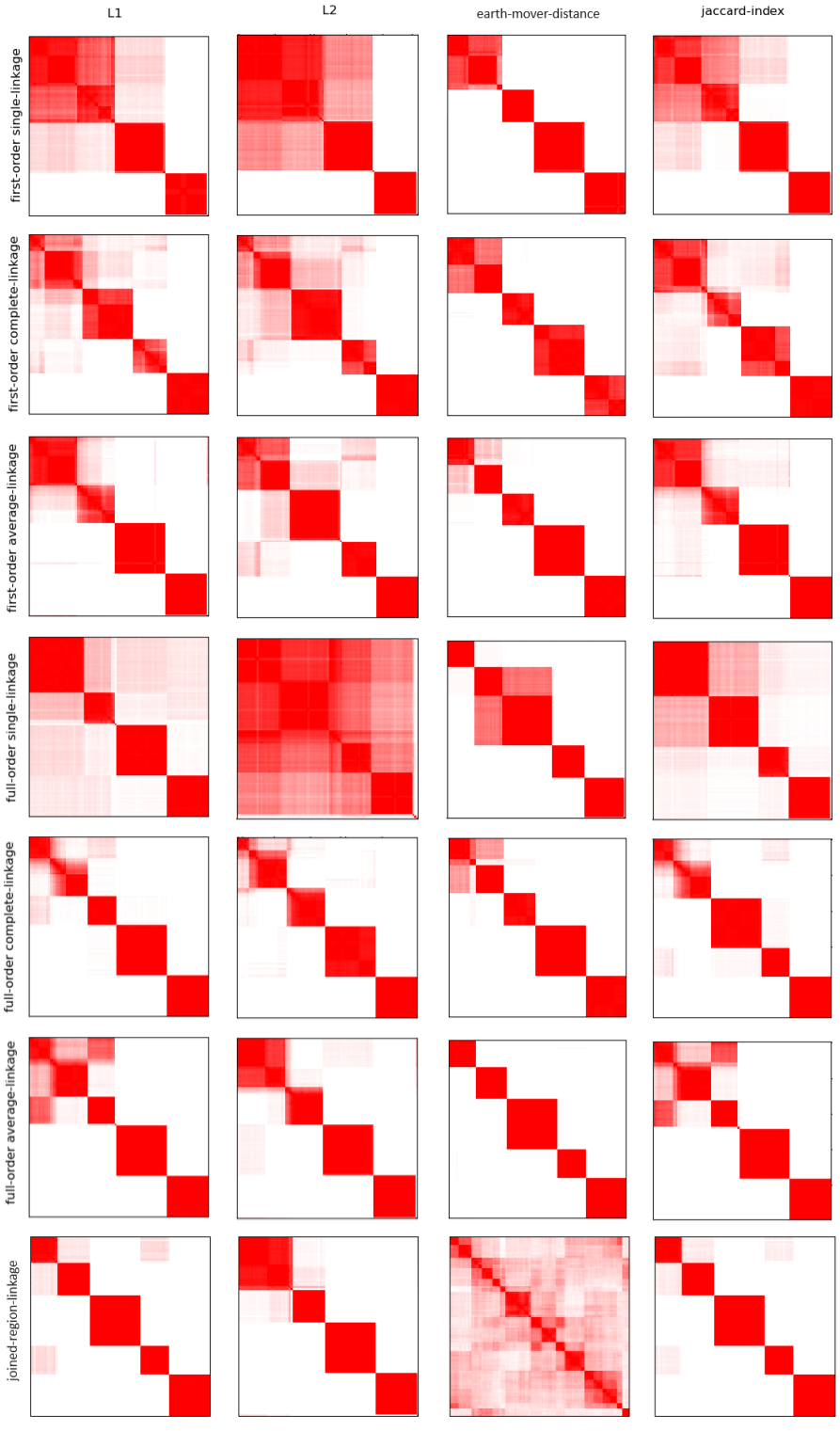



**Figure A.2:** Consensus plot for a range of k-values

This fails because of the massive difference in nodes between the suggested regions. When $k < 5$ the regions with less nodes are split up, causing a small increase in the area under the curve, and when $k > 5$, it affects the regions with more nodes, causing larger differences.

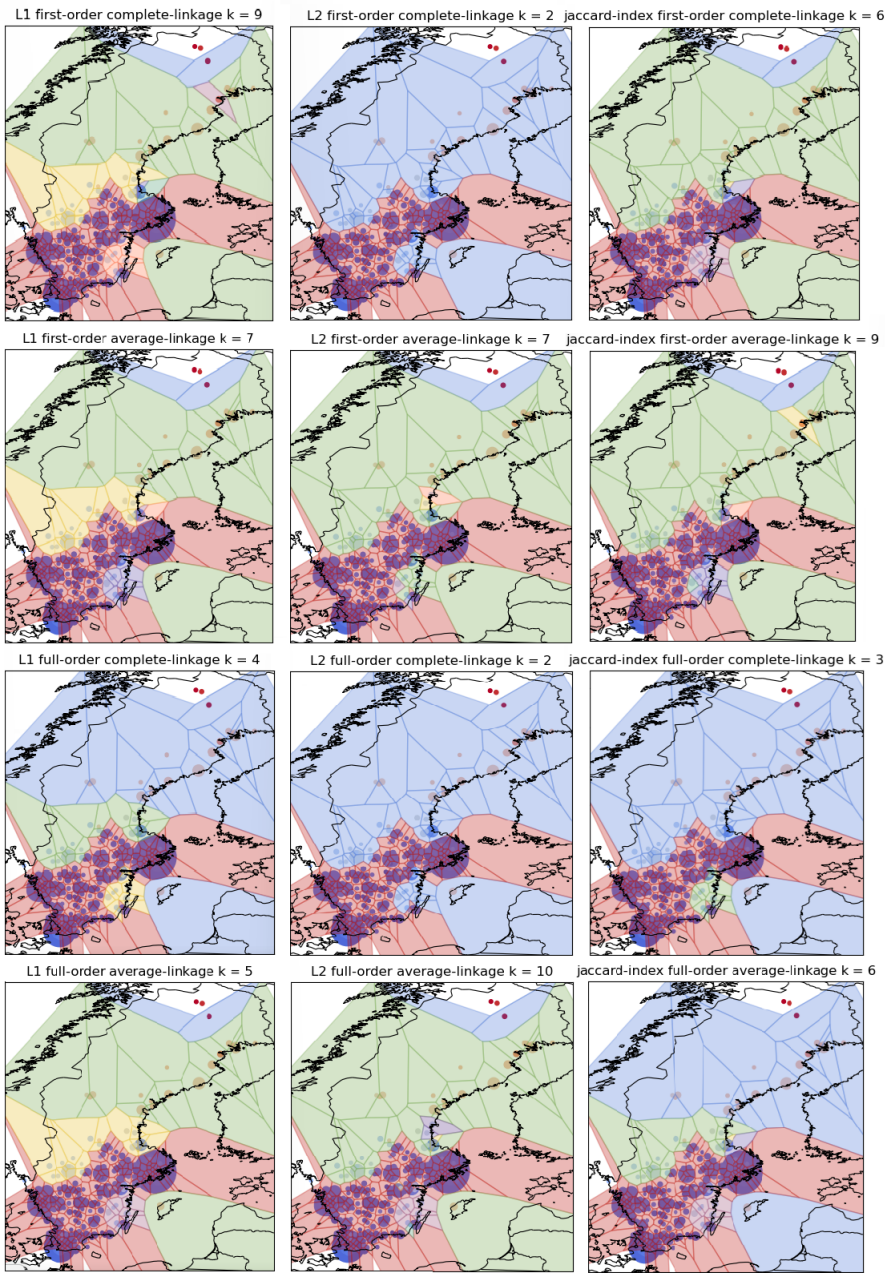
# B

## Confusion matrices for data set 2, $k = 5$



# C

# Individual partitionings from 12 REDCAP setting combinations on data set 4



**DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY