UNIVERSITÀ DEGLI STUDI DI PISA
Facoltà di Scienze, Matematiche, Fisiche e Naturali
Facoltà di Economia
Corso di laurea specilistica in Informatica per l'Economia per l'Azienda

TESI DI LAUREA
TITOLO

SCOT: SPATIAL CLUSTERING OF GERMAN TOWNS

RELATORE
Prof.ssa Fosca GIANNOTTI

RELATORE ECONOMIA
Prof. Nicola Ciaramella

Candidato
Andrea ZANDA

ANNO ACCADEMICO 2006-07

Contents

Abstract

The GIS revolution and the increasing availability of GIS databases emphasize the need to better understand the typically large amounts of spatial data. Clustering is a fundamental task in Spatial Data Mining and many contributions from researchers in the field of Knowledge Discovery are proposing solutions for class identification in spatial databases. The term spatial data refers to a collection of (similar) spatial objects, e.g. areas, lines or points. In addition to geographic information, each object also possesses non-spatial attributes. In order to apply traditional data mining algorithms to such data, the spatial structure ans relational properties must be made explicit. SCOT deals with the special case of grouping German towns. The towns are related to each other by the various streets connecting them. Each town also possesses an inner spatial structure, the local street network, and further non-spatial information. This thesis considers all three kinds of information for the clustering of towns. It exploits the concept of neighborhood to capture relational constraints, measures the similarity of the structures of local street networks and transforms the most important non-spatial attributes. SCOT is part of a project at Fraunhofer IAIS, Germany, and has been successfully applied in practice.

Introduction

The emergence of geographical Information Systems (GIS) with convenient and affordable methods for storing large amounts of spatial data has accelerated the rate at which sheer quantity of spatially referenced information is collected in electronic and magnetic media. It is argued that the GIS revolution and the increasing availability of GIS databases emphasizes the need for exploratory (discovery) rather than confirmatory methods of analysis [16]. Spatial Data Mining [14] is the discovery of interesting relationships and characteristic that may exist implicitly in spatial databases. Data mining in spatial databases aims at a) extracting interesting spatial patterns and features, b) capturing intrinsic relationships between spatial and non- spatial data, c) presenting data regularity concisely and at higher conceptual levels, and d) helping to reorganize spatial databases to accommodate data semantics and to achieve better performance. Current research seeks techniques for artificial searches that are able to hunt out localized patterns or database anomalies in geographically referenced data by reducing the need for direction ("where" to look or "what" to look for). In the following we describe the context of this thesis and formulate the *problem setting*. Fraunhofer IAIS developed within an industry project a *frequency map* for towns with more than 50.000 inhabitants (*big towns*) in Germany. A *frequency map* is a map layer which states for each street segment the average number of passing vehicles and pedestrians per hour. It is impossible to take this frequency for all streets in Germany, in fact only a few were measured in loco, the others were inferred by statistical models. Therefore, at the beginning for each of 182 *big towns* a number of frequency measurement for street segments were available. They are been used to infer through

a statistical model the frequencies of comparable street segments in the same town. The goal of the proceeding project was to build a *frequency map* for towns between 10 and 50 thousands inhabitants which will be refered to as *focus towns*.

In Germany, a total number of 2482 focus towns exist. This amount does not allow to measure frequencies at selected places within each city. Therefore, the technique of inferring frequencies from comparable street segments of the same town, as had been done previously, could not be applied in this project. The solution was to infer the frequency of a certain street segment from the frequency of a comparable street segment in another *comparable* town. In consequence, it is necessary to identify similar towns before the inference of traffic frequencies can take place. The task of SCOT is therefore to find groups of comparable towns within Germany.

Clustering is the task of identifying groups in a data set by some natural criteria of similarity. The idea behind the use of clustering of spatially referenced data is that it provides a means of generalization of the spatial component of the data associated with a GIS [4].

What characterizes the spatial data mining is the crucial role played by the implicit relations among the objects [13].The literature in spatial data mining offers interesting *approaches* for the spatial clustering problem.

A solution to mine geo-referenced data for a spatial clustering is to group structured objects, collected at different sites, such that data inside each cluster models the continuity of socio-economic or geographic environment, while separate clusters model variation over the space [11]. This solution did not fit our problem because the objects should have a spatial contiguity to be grouped.

In [15] a method is applied by considering the spatial and the non-spatial data separately and by applying two different approaches: one (named SD-CLARANS) computes the clusters considering first the spatial data and then it characterize the clusters using the non-spatial data; the other (named NSD-CLARANS) extract the spatial clusters from groups of non-spatial data, i.e. the clustering is performed first on the non-spatial attributes of data. The two approaches gives different results

according if we first compute the spatial information or non-spatial. For our purposes the two approaches each with its weight should be complementary in one process and not constraining.

A typical group of algorithms for the clustering of spatial data are density-based algorithms. For example DBSCAN [3], finds arbitrarily shaped clusters of objects that are close in space. The spatial role in the algorithm is not sufficient for our problem. Could be interesting to group towns of a certain area, but it should be not constraining. For example there could be two towns in two different areas of Germany that could be similar.

The approaches presented did not fit our problem so we decided to bring to life SCOT. The similarity between the frequency level of two towns depends on the inner-city structure, the relationships between towns and other non-spatial information as the number of inhabitants. Therefore, SCOT extracts spatial characteristics and relationships explicitly and applies traditional clustering algorithms to the resulting features.

The figure 1 represent the SCOT's workflow. On the left we can see the three main topics touched with this thesis: Urban Geography, Computer Vision and Graph theory. Urban Geography helps us to focus our efforts on what is discriminating to classify towns. For example Urban Geography literature remarks the importance of street networks to characterize a town, there are towns with radial and grid street networks. Computer Vision provides the approaches to represent a spatial object. It answer to the question: How can I represent spatial information? The answer is the Graph Theory which offers a useful data structure in order to compute such information. It supplies the means to handle street networks and relationships between towns. These three topics are discussed in chapter 1, section 1.1, 1.2 and 1.3 respectively.

The figure 1 shows a database containing spatial and non-spatial information. On the middle left this information is used for spatial data preparation. Spatial data preparation is the core of SCOT, this task makes explicit characteristics and relationships of towns. It is threaten in chapter 2 and it is divided in three main

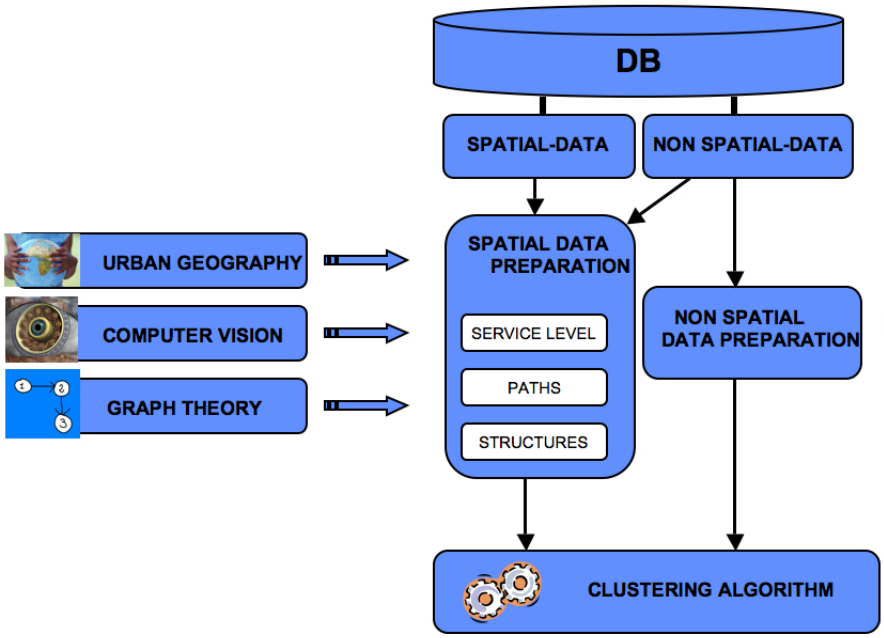


Figure 1: Workflow of the project

sub-tasks: *Service Level*, *Paths* and *Structures*. The goal of task *Service Level* in section 2.3 is to discover for every town the level of services (i. e. goods or public services) which is offered by close towns. In section 2.4 we identify the *focus towns* which lie in a main street (motor-way or federal highway) between two *big towns*. It is interesting because the frequency of such towns could be different from a frequency of a town which does not lie in a main street between two *big towns*. In fact the traffic from a *big town* to another could increase the frequency of the town in between. The third sub-task exploits the town street network similarities. A representation of street network for each *focus town* is given, then they are are compared each other in order to find groups of towns with similar street network.

On the middle right of figure 1 we have non-spatial data preparation, section 2.6, which threat non-georeferenced data, for example points of interest or number of houses in each town.

The final application of clustering algorithms is presented in chapter 3. The results

obtained from spatial and non-spatial data preparation are merged and given in input to traditional clustering algorithms, bottom part of figure 1.

Chapter 1

RELATED WORK

1.1 Urban Geography

Cluster Analysis cannot prescind from a basis-knowledge of domain. In this chapter a brief summary of urban form and structure study in Germany [8] will be shown, pointing out what is considered relevant for the analysis in the specific domain.

1.1.1 Main Studies

When geography became established as a scientific discipline in German universities in the 1880s, the undertaken problems were concerned with the two basic questions of where and why urban places had come into existence. A good example of the methodology of this generation of geographers was the Ratzel's treatise [17]. With the turn of the century the dominant topic changed modifying the focus on the layout of urban places, the street patterns, transportation lines, squares, open spaces and the three-dimensional building fabric. So the following three decades became known as the morphological or physionomic ephoc of German urban geography.

As to the methodology of urban geographers during the morphological epoch, the town plan became the characteristic instrument of their endeavours. The scientists

underlined the importance of historical town plans for tracing the original settlement layout.

The various publications on the layout of German towns stressed two issues that initiated a vigorous discussion among German geographers. First, by means of comparison of the layout of towns in various parts of Germany some authors came to the conclusion that not only rural villages but also towns looked different on either side of the so called Elbe-Saale-Line. These two rivers were, for many centuries, the dividing line between the regions to the west, settled by Germanic tribes, and the regions to the east, settled by Slavic tribes and only after 1200 colonized by Germanic people from farther west. Consequently, towns to the west of the Elbe-Saale-Line were believed to have grown over a number of centuries with the result that their street patterns were more or less irregular. In contrast, the towns east of the Elbe-Saale-Line were founded by the colonizing people under the rule of particular governing authorities on the basis of some prepared plan so that they had a much more regular street pattern, if not an exact grid. The second issue was the significance of market places and town walls in the layout of towns. It was argued that the market had developed over several centuries from a mere widening of the main street to a centrally-located square of increasing size, and in some towns there were even several market places each devoted to the trade in a particular commodity, such as horses or other animals, meat, grain, vegetables, fish or forest products. There was, indeed, certain evidence of the market place becoming more prominent with increasing distance east of the Elbe-Saale-Line.

Almost every German town that existed by 1200, or was founded after 1200, was a walled town. In many cases the wall followed a roughly circular line, and this had an impact on the direction of at least a few streets. Some streets ran parallel to the wall while others ended in front of it. One or two thoroughfares were oriented toward the gates, which were the only entry points into town.

When, after 1500, a number of towns were fortified with large ramparts and bastions, these fortifications had a still greater impact on the layout of towns. Some newly-founded fortresses, such as Neuf Brisach near the French-German border, had

a spectacular layout dominated by a huge centrally-located place des armes and an exact grid pattern of streets. When in more recent years those fortifications were dismantled, the open spaces were often used for ring roads and railway lines.

A special category of towns in Germany were the numerous court-towns of the royalty and high nobility of the former German sovereign territories. These towns used to be designed according to the founder's conception, the streets being oriented toward the royal palace. Some such layouts were a combination of both a radial street pattern and a grid.

The period from 1928 to the mid-1950s was dominated by research on urban functions and urban structure. Christaller's Central place theory of 1933 is a milestone; it plays a central role in this thesis and a detailed description is supplied in section 1.1.2.

Lafrenz [10] used the city of Lübeck as an example for what he called Bewertungszyklen (evaluation of cycles of buildings). According to his findings there have been two such cycles in this town's recent past. The economic growth during the nineteenth century led to lot sizes, street widths and building heights that differed from the traditional street pattern and building fabric of the old town. From the beginning of the twentieth century through to the 1920s, people made attempts to correct those errors. A second cycle started after the Second World War when, during the course of reconstruction, many old and partially destroyed buildings were torn down and replaced by modern structures, these usually being out of proportion to the traditional building stock. After 1970, the urban conservation movement made people sensitive to such blunders, and another revaluation in favour of traditional forms led to a more subtle treatment of townscapes.

Although urban form has not, in recent years, received adequate attention from German geographers.

1.1.2 Central Place Theory

Central Place Theory [19] attempts to explain the spatial arrangement, size, and number of settlements. By examining and defining the functions of the settlement structure and the size of the hinterland, Christaller found it possible to model the pattern of settlement locations using geometric shapes. Christaller made a number of assumptions such as: all areas have an isotropic (all flat) surface, an evenly distributed population, evenly distributed resources, the purchasing power of all consumers is similar and consumers will patronize the nearest market, transportation costs equal in all directions and are proportional to distance, there are no excess profits (perfect competition).

The theory consists of two basic concepts, threshold and range. The threshold is the minimum population that is required to bring about the provision of certain good or services. The range of good or services is the average maximum distance people will travel to purchase goods and services. From these two concepts, see figure 1.1, the lower and upper limits of goods or services can be found. With the upper and the lower limits, it is possible to see how the central places are arranged in an imaginary area.

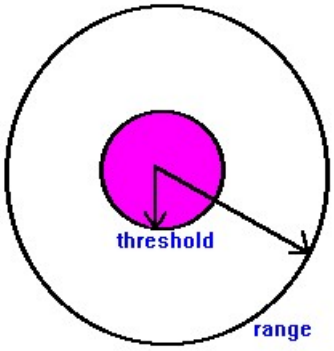


Figure 1.1: Christaller's Central Place Theory - Threshold and Range

The central place hierarchy is defined from Christaller as shown in table 1.1

However, the circular shape of the market areas results in either un-served areas or

Towns hierarchy	Population	Radius
Marktort	1000	4.0
Amtsort	2000	6.9
Kreissstadt	4000	12
Bezirkstadt	10000	20.7
Gaustadt	30000	36
Provinzstadt	100000	62.1
Landstadt	500000	108

Table 1.1: Christaller's Central Place Hierarchy

over-served areas. To solve this problem, Christaller suggested the hexagonal shape of the markets. Within a given area there will be fewer high order cities and towns in relation to the lower order villages and hamlets. For any given order, theoretically, the settlements will be equidistance from each other. The higher order settlements will be further apart than the lower order ones.

Christaller noted three different arrangements of central places according to: the marketing principle, the transportation principle and the administrative principle. According to the transport principle, the central places would thus be lined up on straight traffic routes which fan out from the central point and the lower order centers are located at the midpoint of each side of a hexagon. Central places are nested as shown in Figure 1.2.

The market area of a higher-order place includes a half of the market area of each of the six neighbouring lower-order places, as they are located on the edges of hexagons around the high-order settlements. This generates a hierarchy of central places which results in the most efficient transport network.

The theory does a reasonably good job of describing the spatial pattern of urbanization. No other economic theory explains why there is a hierarchy of urban centers. How the theory helps us to better define town typologies according to their market influences, will be explained in chapter 2.3.

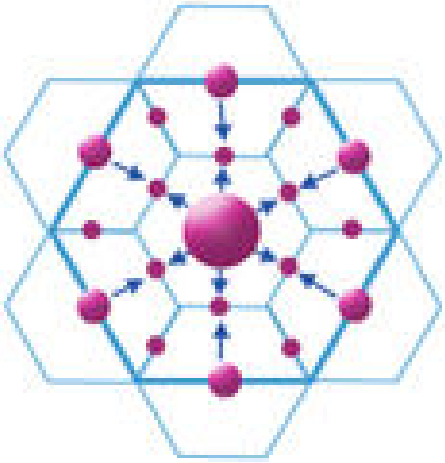


Figure 1.2: Christaller's Central Place Theory - Transportation Principle

1.2 Computer Vision

Computer vision is the science and technology of machines that see. As a scientific discipline, computer vision is concerned with the theory and technology for building artificial systems that obtain information from images [20].

Object recognition is one of the most important aspects of computer vision. In order to recognize and identify objects, the vision system must have one or more stored models of the objects that may appear in the universe it deals with. This chapter discusses the object models used by vision systems and the matching procedures used for recognizing objects. The topic covered is the two-dimensional models. This section has an important role, it helps us to represent the spatial information, local street networks and relationships between towns.

1.2.1 Two-dimensional Object Representation

Two-dimensional object representation and recognition by computers began in the 1960s and has been an active area of research ever since. Two-dimensional shape analysis is useful in a number of applications of machine vision, including medical image analysis, aerial image analysis, and manufacturing. The method used for shape recognition often depends on the particular representation selected. Thus we begin by looking a the various representations that have been used. They fall loosely into four classes: representation by global features, by local features, by boundary description and by skeleton. We will present them in this section based on [7].

Global Feature Representation

A two-dimensional object can be thought of as a binary image. The pixel of the object have value 1, and the pixels outside the object have value 0. Because of this relationship, it is natural to represent shapes by using some of the features which are used for representing binary images. Commonly used features for two-dimensional

shape representation include area, perimeter, moments, circularity, and elongation. Some of the earliest shape recognition work utilized moments and Fourier descriptors.

Fourier descriptors provide a meaning for extracting global features from two-dimensional shapes. Rather than characterizing the entire area of the shape, Fourier descriptors usually are defined to characterize the boundary. The main idea is to represent the boundary as a function of one variable $\phi(t)$, expand $\phi(t)$ in its Fourier series, and use the coefficients of the series as Fourier descriptors (FDs). A finite number of these FDs can be used to describe the shape.

There have been several different suggestions for defining ϕ and constructing the FDs. We follow Persoon and Fu in their modification of Granlund's FD definition for curves represented by polygons, the most common representation in computer vision. Let y be a clockwise-oriented, simple closed curve represented by the parameterized function

$$Z(l) = [x(l), y(l)], \quad 0 \leq l \leq L \tag{1.1}$$

where l is the arc length along γ. A point moving along the curve generates the complex function

$$u(l) = x(l) + jy(l) \tag{1.2}$$

which is a periodic function with period L. The Fourier series expansion of $u(l)$ is given by

$$u(l) = \sum_{-\infty}^{\infty} a_n e^{jn(2\pi/L)^l} \tag{1.3}$$

The FDs are the coefficients $\{a_n\}$ defined by

$$a_n = \frac{1}{L} \int_0^L u(l) e^{-j(2\pi/L)^{nl}} \tag{1.4}$$

Persoon and Fu assume that the two-dimensional shape to be described is represented as a sequence of m points $< V_o, V_1, ..., V_m = V_o >$. From the sequence of points, they define a sequence of unit vectors

$$b_k = \frac{V_{k+1} - Vk}{|V_{k+1} - V_k|} \tag{1.5}$$

and a sequence of cumulative differences

$$l_k = \sum_{i=1}^{k} |V_i - V_{i-1}|, \quad k > 0 \tag{1.6}$$

The FDs are then defined by

$$a_n = \frac{1}{L(\frac{n1\pi}{L})^2} \sum_{k=1}^{m} (b_{k-1} - b_k) e^{-j(2\pi/L)^{nl}} \tag{1.7}$$

Using these FDs, Persoon and Fu defined a distance measure to be used in shape comparison. Suppose α and β are two curves to be compared and that $\{a_n\}$ is the sequence of FDs of α, $\{\beta_n\}$ is the sequence of FDs of β, and M is the number of harmonics used. Then the distance measure is given by

$$d(\alpha, \beta) = \left[\sum_{n=-M}^{M} |a_n - b_n|^2 \right]^{\frac{1}{2}} \tag{1.8}$$

In order to compare an unknown curve α to a model curve β , they developed a numeric procedure that solves for the scale, rotation, and starting point that minimizes $d(\alpha, \beta)$. The resultant distance is a measure of the similarity between α and β. Profitt (1982) discusses a normalization technique that can be used in conjunction with FD representation. Chellappa and Bagdazin (1984), using an autoregressive model, obtain estimates of the variances of the Fourier coefficients. Lin and Chellappa (1987) give a procedure for estimating the Fourier coefficients under the constraint of a known value for $perimeter^2/area$. This improves the estimate even when some part of the true boundary has been occluded. Stracklee and Nagelkerke (1983) note that

when the shape is represented as the tangent angle of the boundary as a function of arc length, then truncating the Fourier coefficient representation can produce a representation in which the reconstructed boundary does not close on itself. They give a procedure for the estimation of the Fourier coefficient representation of the tangent angle function that guarantees the reconstructed boundary will close on itself.

Local Feature Representation

A two-dimensional object can also be characterized by its local features, their attributes, and their interrelationships. The most commonly used local features in industrial-part recognition are holes and corners. Holes can be detected by a connected component procedure followed by boundary tracing or, if the shapes of the holes are known in advance, through the operations of binary mathematical morphology. Corner detection can be performed on a binary image or on a gray tone image. Local features must be organized into some type of structure for matching. The most common type of structure is a graph whose nodes represent local features and their properties or measurements and whose edges represent relationships among the features. For example, if the features are all corners, then the angle at which the lines meet is a feature property, and each corner can be related to its two adjacent one spatial relationship that is the obvious one to use for describing the relationships among holes or among holes and corners. Distance from hole to hole or from hole to corner is one possibility. If the object will not be occluded, then the centroid can be used as a focal point and the positions of all other features expressed in relation to the position of the centroid.

A feature-based system for recognition of industrial parts that uses both holes and corners was developed by Bolles and Cain (1982). Their local-feature-focus method finds one key feature, the focus feature, in an image and uses it to predict a few nearby features to look for. It uses graph matching to find the largest cluster of image features matching a cluster of object features near the focus feature. Once such a cluster is found, a hypothesis verification procedure adds more features and also checks the boundary of the object. The features used in their example system were regions that

had properties of intensity (black or white), area-to-axis ratio, and corners in which the size of the included angle was the measured property. The system's knowledge of the most important features to look for in each object and of a cluster of other features that should stand in certain relationships to the focus feature was generated automatically from training images. The training program identified similar local features in different objects, computed symmetries, marked structurally equivalent features that could not be distinguished locally, built feature-centered descriptions, selected nearby features, and ranked the focus features according to the size of the graph that would have to be matched.

Boundary Representation

Boundary representation is the most common representation for two-dimensional objects. There are three main ways to represent the boundary of an object: as a sequence of points, by its chain code, and as a sequence of line segments.

If the *Boundary as a Sequence of Points* is represented as the points of the boundary generally come from some kind of border-following or edge-tracking algorithm performed on a digital image. The result of such an operation is a list of pixel coordinates. The list can be maintained as a whole, converted into one of the other two main boundary representations, or processed to produce a smaller list of interest points. Interest points are points on the boundary that have some special property that makes them useful in a given matching algorithm. The affine-invariant matching algorithm, defined later in this chapter, requires a set of interest points that are described as being sharp convexities or deep concavities of the boundary of the shape. One method of extracting these interest points from the original sequence of boundary points of the curve is the curve-partitioning algorithm described in Phillips and Rosenfeld (1987). Given a point P on the curve and a fixed arc length k, that includes P, and let $M(P, C)$ be the maximum distance from P to all such chords. P is a partitioning point of the curve, if the value of $M(P, C)$ is a local maximum (for the given k) and also exceeds a threshold $t(k)$. This method finds points of high curvature along the boundary. It can be modified to select a point P that is the median point

in a sequence of points $< P_1, ..., P_n >$ for which $M(P_i, C), i = 1, ..., n$ are all local maxima. In this way it detects not only very sharp corners but also points of high curvature along the boundary that are part of a section of approximately constant curvature.

The *Chain Code Representation* says that an ordered list of the coordinates of boundary pixels is a sufficient representation for any boundary, however, it may be too fine a representation for many applications. Freeman (1961, 1974) developed a representation called chain encoding that can be used at any level of quantization and that saves space required for the row and column coordinates. A boundary (or any curve) to be encoded is first quantized by placing over it a square grid whose side length determines the resolution of the encoding.

The *Boundary as a Sequence of Line Segments* is the third common representation for the boundary of a two-dimensional shape. Although any sequence of points can be thought of its a sequence of line segments, this representation is generally used after the original sequence of boundary points has been segmented into a set of line segments representing near-linear portions of the boundary. Pavlidis' split-and-merge algorithm (Pavlidis and Horowitz, 1974) is one possible way to achieve such segmentation. Fitting line segments to the clusters of adjacent collinear points detected by a Hough transform or grouped together by a line-finding procedure such as the Burns line detector (Burns, Hanson, and Riseman, 1986) is another possibility. Once the sequence of line segments has been computed by some method, it can be converted into a model of the shape that can be used in shape recognition or other matching tasks. A model for representing and matching sequences of line segments was given by Davis (1979). Davis represented a line segment sequence by the sequence of junction points $< X_i, Y_i, \alpha_i >$ where a pair of lines meet at coordinate location (X_i, Y_i) with angle magnitude α_i. Given a sequence $O = 0_1, 0_2, ..., 0_n$ of junction points representing the boundary of a model object 0 and a similar sequence $T = T_2, T_2, ..., T_n$ representing the boundary of a test object T, the goal is to find an association $F : \{1, 2, ..., m\} \rightarrow \{1, 2, ..., n\} \cup \{missing\}$ that satisfies $i < j \rightarrow F(i) < F(j)$ or either $F(i) = missing$ or $F(j) = missing$. Davis used con-

straints on both sides (line segments) and angles to define what is meant by a best mapping for this problem. Let $M(i, j)$ be a local evaluation function that measures the goodness of the match of junction i of T to junction j of 0, based on the difference between the angles α_i and α_j. Let $S_{ij}(i', j')$ be a measure of the consistency of mapping junction i to junction i' and junction j to junction j', based on the difference between the segment lengths of T_iT_j and $O_{i'}O_{j'}$. The cost of a mapping F is given by

$$C(F) = \sum_{i=1}^{m} M\left[i, F(i)\right] + \sum_{i=1}^{m}\sum_{j=1}^{m} S_{ij}\left[F(i), F(j)\right] + P(m_T) + P(m_O) \tag{1.9}$$

where P is a penalty function for missing angles.

Skeleton Representation

Although the boundary of a two-dimensional object gives full information on the shape of the object, this may not be the most suitable information for matching. Particularly for shapes that can be thought of as a union of long, sometimes thin parts called strokes, the essence of the shape can be described as a sequence of the line segments that capture the linearity of the strokes. Blum (1973) and Blum and Nagel (1978) defined the symmetric axis transform of a two dimensional object as the set of maximal circular disks that fit inside the object. The object can b represented by its symmetric axis (the locus of the centers of these maximal disks) plus the set of distances of these centers to the boundary of the object. The symmetric axis is one example of a skeleton description of a two-dimensional object. The symmetric axis is not always completely representative of the strokes of an object. Notice that the symmetric axis of a rectangle, rather than being a single line, consists of five line segments. This property and the fact that the symmetric axis is extremely sensitive to noise make it difficult to use in matching.

1.3 Graph Theory

1.3.1 Graph definitions

A graph is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are 2-element subsets of V. To avoid notational ambiguities, we shall always assume tacitly that $V \cap E = 0$. The elements of V are the vertices (or nodes, or points) of the graph G, the elements of E are its vertex edges (or lines). The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge. Just how these dots and lines are drawn is considered irrelevant: all that matters is the information of which pairs of vertices form an edge and which do not. An example of Graph is given in Figure 1.3.1 where $V = \{1, ..., 7\}$ and edges set $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}, \{5, 7\}\}$.

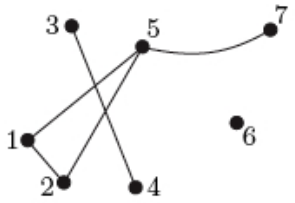


Figure 1.3: Graph example

A graph with vertex set V is said to be a graph on V. The vertex set of a graph G is referred to as $V(G)$, its edge set as $E(G)$. These $V(G)$, $E(G)$ conventions are independent of any actual names of these two sets: the vertex set W of a graph $H = (W, F)$ is still referred to as $V(H)$, not as $W(H)$. We shall not always distinguish strictly between a graph and its vertex or edge set. For example, we may speak of a vertex $v \in G$ (rather than $v \in V(G)$), an edge $e \in G$, and so on [2].

We have a *labeled graph* when a vertex labeling function or an edge labeling function is associated. Given a graph $G := (V, E)$ such that V is the set of vertices and E is the set of edges, a vertex labeling is a function from some subset of the integers to

the vertices of the graph. Likewise, an edge labeling is a function from some subset of the integers to the edges of the graph [22].

A *weighted graph* associates a label (weight) with every edge in the graph. Weights are usually real numbers. They may be restricted to rational numbers or integers. Certain algorithms require further restrictions on weights; for instance, the Dijkstra algorithm works properly only for positive weights. The weight of a path or the weight of a tree in a weighted graph is the sum of the weights of the selected edges. Sometimes a non-edge is labeled by a special weight representing infinity.

1.3.2 Paths

A path is a non-empty graph $P = (V, E)$ of the form $V = \{x_0, x_1, ..., x_k\}$ and $E = \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\}$, where the x_i are all distinct. The vertices x_0 and x_k are linked by P and are called its ends; the vertices $x_1, ..., x_{k-1}$ are the inner vertices of P. The number of edges of a path is its *length*, and the path of k is length denoted by P^k. Note that k is allowed to be zero. We often refer to a path by the natural sequence of its vertices, writing, say, $P = x_0x_1...x_k$ and calling P a path from x_0 to x_k (as well as between x_0 and x_k)[2].

Depht-First-Search

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph [23]. Intuitively, one starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it had not finished exploring. In a

non-recursive implementation, all freshly expanded nodes are added to a LIFO stack for exploration.

Time complexity is proportional to the number of vertices plus the number of edges in the graph it traverses ($O(|V| + |E|)$).

When searching large graphs that cannot be fully contained in memory, DFS suffers from non-termination when the length of a path in the search tree is infinite. The simple solution of "remember which nodes I have already seen" does not always work because there can be insufficient memory. This can be solved by maintaining an increasing limit on the depth of the tree, which is called iterative deepening depth-first search.

Dijkstra's algorithm

Dijkstra's algorithm, named after its discoverer, the dutch computer scientist Edsger Dijkstra, is a greedy algorithm that solves the single-source shortest path problem for a directed graph with non negative edge weights [21].

For example, if the vertices (nodes) of the graph represent cities and edge weights represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between two cities.

The input of the algorithm consists of a weighted directed graph G and a source vertex s in G. We will denote V the set of all vertices in the graph G. Each edge of the graph is an ordered pair of vertices (u, v) representing a connection from vertex u to vertex v. The set of all edges is denoted E. Weights of edges are given by a weight function $w : E \to [0, \infty)$; therefore $w(u, v)$ is the cost of moving directly from vertex u to vertex v. The cost of an edge can be thought of as (a generalization of) the distance between those two vertices. The cost of a path between two vertices is the sum of costs of the edges in that path. For a given pair of vertices s and t in V, the algorithm finds the path from s to t with lowest cost (i.e. the shortest path). It can also be used for finding costs of shortest paths from a single vertex s to all other vertices in the graph.

In figure 1.3.2, $u := extract\_min(Q)$ searches for the vertex u in the vertex set Q that has the least $dist[u]$ value. That vertex is removed from the set Q and returned to the user. $length(u, v)$ calculates the length between the two neighbor-nodes u and v. alt on line 10 is the length of the path from the root node to the neighbor node v if it were to go through u. If this path is shorter than the current shortest path recorded for v, that current path is replaced with this alt path.

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:           // Initializations
 3          dist[v] := infinity               // Unknown distance function from s to v
 4          previous[v] := undefined
 5      dist[source] := 0                     // Distance from s to s
 6      Q := copy(Graph)                      // Set of all unvisited vertices
 7      while Q is not empty:                 // The main loop
 8          u := extract_min(Q)               // Remove best vertex from priority queue
 9          for each neighbor v of u:
10              alt = dist[u] + length(u, v)
11              if alt < dist[v]              // Relax (u,v)
12                  dist[v] := alt
13                  previous[v] := u
```

Figure 1.4: Dijkstra's algorithm pseudocode

If we are only interested in a shortest path between vertices source and target, we can terminate the search at line 9 if $u = target$. Now we can read the shortest path from source to target by the iteration in figure 1.3.2.

```
1 S := empty sequence
2 u := target
3 while defined previous[u]
4     insert u at the beginning of S
5     u := previous[u]
```

Figure 1.5: Dijkstra's algorithm pseudocode

Now sequence S is the list of vertices constituting one of the shortest paths from source to target, or the empty sequence if no path exists.

Here we discuss the relational distance as a framework for matching.

1.3.3 Graph matching

Given a graph $G = (V, E)$, a matching M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex [18].

We say that a vertex is matched if it is incident to an edge in the matching. Otherwise the vertex is unmatched.

A maximum matching is a matching that contains the largest possible number of edges. There may be many maximum matchings. The matching number of a graph is the size of a maximum matching. A maximal matching is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching, that is, M is maximal if it is not a proper subset of any other matching in graph G. In other words, a matching M of a graph G is maximal if every edge in G has a non-empty intersection with at least one edge in M. Note that every maximum matching must be maximal, but not every maximal matching must be maximum.

A perfect matching is a matching which covers all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. Every perfect matching is maximum and hence maximal. In some literature, the term complete matching is used.

In most case the matching is not complete, so we need a measure which indicate how a graph matches another, in the following we define the concept of Relational-Distance.

Relational-Distance Definition

A *relational description* D_x is a sequence of relations $D_x = R_1, ..., RI$, where for each $i = 1, ..., I$, there exist a positive integer n_i with $R_i \subseteq X^{ni}$ for some set X. Intuitively X is a set of the parts of the entity being described, and the relations R_i indicate various relationships among the parts [7].

Let $D_a = R_1, ..., R_I$ be a relational description with part set A and $D_b = S_1, ..., S_L$ a relational description with part set B.

Let f be any one-one, onto mapping from A to B. For $R \subseteq A^N$, N a positive integer, the *composition* $R \circ f$ of relation R with function f is given by

$$R \circ f = \left\{ (b_1, ..., b_N) \subseteq B^N \mid \textit{there exist } (a_1, ..., a_N) \subseteq R \textit{ with } f(a_n) = b_n, n = 1, ..., N \right\}$$

This composition operator takes N-tuples of R and maps them, component by component, into N-tuples of B^n.

The function f maps parts from set A to parts from set B. The *relational distance* of f for the relations (S_I and S_L) in D_A and D_B is given by

$$RD(A, B) = \frac{I - |(R - (S \circ f^{-1})|}{I} + \frac{L - |(S - (R \circ f)|}{L}$$

The *relational distance* indicates a value according how many tuple in R match with f the tuples in S and how many tuples in S match with f^{-1} the tuples in R .

1.4 Data Mining and Knowledge Discovery

This chapter has the goal to present the concept of data mining and the state-of-art techniques we are going to use in the next chapters. Witten and Frank in [24] offer a complete panoramic on data mining practical tools and Miller and Hanh [12] discusses the process of geographic data mining. In the first subsection we present non-spatial data mining techniques, the second subsection focuses on the case in which the data is geo-referenced.

Data mining is only one step of the knowledge discovery from databases (KDD) process. Data mining involves the application of techniques for distilling data into information or facts implied by the data. KDD is the higher level process of obtaining facts through data mining and distilling this information into knowledge or ideas and beliefs about the mini- world described by the data. This generally requires a human-level intelligence to guide the process and interpret the results based on pre-existing knowledge [12]. The KDD process typically involves the following major steps: background knowledge, data pre-processing, data mining and knowledge construction. So data mining as the part of KDD process has been defined as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" [5] and "the science of extracting useful information from large data sets or databases" [6]. The data mining topic in which we are interested in Clustering, the identification of heterogenous groups that contain objects with similar characteristics. In section 1.4.1 we present partitioning clustering algorithms and in 1.4.2 Agglomrative Hierarchical clustering.

1.4.1 Partitioning Clustering

Given a data set D of n objects in a d-dimensional space, and an input parameter k, a partitioning algorithm organizes the objects into k clusters such that the total deviation of each object from its cluster centre, or from a cluster distribution, is minimized. The deviation of a point can be computed differently in different algorithms

and is more commonly called a *similarity function*. We will present two partitioning algorithms which have different ways to represent their clusters: *K-means* and *EM*. Despite the difference in the representation of the clusters, the two partitioning algorithms share the same general approach when computing their solutions. To see the similarity, we first observe that the two algorithms are effectively trying to find the k centers or distributions that will optimize an objective criterion. Once the optimal k centers or distributions are found, the membership of the n objects within the k clusters are automatically determined. However, to find the global optimal k centers or k distributions is known to be NP-hard (Garey and Johnson 1979). Instead the two algorithms adopt an iterative relocation technique which will find a local optimal. This technique is shown in figure 1.4.1. The two algorithms, however, differ in the criterion function and in the way they handle steps 3 and 4 of the algorithm.

Algorithm 2.1 (Iterative relocation)

Input: The number of clusters k, and a database containing n objects.
Output: A set of k clusters which minimizes a criterion function E.
Method: (1) arbitrary choose k centers/distributions as the initial solution;
(2) repeat
(3) (re)compute membership of the objects;
(4) update some/all cluster centers/distributions according to new membership of the objects;
(5) until (no change to E);

Figure 1.6: Generalized iterative relocation

K-MEANS

The *K-means* algorithm (MacQueen 1967) uses the mean value of the objects in a cluster as the cluster centre. The objective-criterion used in the algorithm is typically the squared-error function defined as

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} |x - m_i|^2 \tag{1.10}$$

where x is an object which belongs to cluster C_i, and m_i is the mean of cluster C_i. The *K-means* algorithm basically follows the structure of the algorithm in figure 1.4.1. In step 3 of the algorithm, *K-means* assigns each object to its nearest centre, forming a new set of clusters. In step 4, all centers of these new clusters are then computed by taking the mean of all objects in each cluster. This is repeated until the criterion function E does not change after an iteration.

The *K-means* algorithm is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k << n$ and $t << n$. The method often terminates at a local optimum.

Besides the general weakness of partitioning-based algorithm, the *K-means* algorithms is also very sensitive to noise and outlier data points, since a small number of such data can substantially influence the mean value [9].

EM

Insted of representing each cluster using a single point, the *EM* algorithm represent each cluster using a probability distribution. Typically, the Gaussian probability distribution is used because according to density estimation theory, any density distribution can be effectively approximated by a mixture of Gaussians (Scott 1992; Silverman 1986). A d-dimensional Gaussian distribution representing a cluster C_i is parametrized by the mean of the cluster μ_i, and $d * d$ covariance matrix M_i. Given a cluster distribution for C_i, the probability of an object occurring at location x is denoted as $P(x|i)$ where

$$P(x|i) = \frac{1}{\sqrt{(2)\pi^d |M_i|}} e^{(1/2)(x-\mu_i)^T (M_i)^{-1}(x-\mu_i)} \tag{1.11}$$

where the superscript T indicates the transpose to a row vector, $|M_i|$ is the determinant of M_i and M_i^{-1} is its matrix inverse. By combining the effect of the different cluster distributions at x, the mixture model probability density function will be:

$$P(x) = \sum_{k}^{i=1} W_i P(x|i) \tag{1.12}$$

where W_i is the fraction of the database represented by C_i. Unlike the *K-means*, an object in *EM* clustering can be a member of each of the k clusters with different probabilities of membership. Probability of an object at x belonging to cluster C_i can be computed as:

$$P(i|x) = W_i \frac{P(x|i)}{P(x)} \tag{1.13}$$

Referring back to the algorithm in figure 1.4.1, after the random initialization of the model, the *EM* algorithm will compute the membership of each object in step 3 by applying formulae 1.11, 1.12 and 1.13. the new values of W_i, μ_i and M_i are then computed in step 4 using the following formulae:

$$W_i = \frac{1}{n} \sum_{x \in D} P(i|x) \tag{1.14}$$

$$\mu_i = \frac{\sum_{x \in D} x P(i|x)}{\sum_{x \in D} P(i|x)} \tag{1.15}$$

$$M_i = \frac{\sum_{x \in D} P(i|x)(x - \mu_i)(x - \mu_i)^T}{\sum_{x \in D} P(i|x)} \tag{1.16}$$

As a criterion function, *EM* clustering tries to maximize the log likelihood of the mixture model computed as

$$E = \sum_{x \in D} log(P(x)) \tag{1.17}$$

.

When the increase in the log likelihood between two successive iterations is negligible, the algorithm terminates [9].

1.4.2 Agglomerative Hierarchical Clustering

Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

- Agglomerative: Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.
- Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

Agglomerative hierarchical clustering techniques are by far the most common, and, in this section, we will focus exclusively on these methods. A hierarchical clustering is often displayed graphically using a tree-like diagram called a dendrogram, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). Figure 1.4.2 shows an example of a dendogram for a set of four two-dimensional points.

Many agglomerative hierarchical clustering techniques are variations on a single approach: starting with individual points as clusters, successively merge the two closest clusters until only one cluster remains. This approach is expressed more formally in the algorithm in figure 1.4.2.

The key operation of algorithm 8.3 is the computation of the proximity between two clusters, and it is the definition of cluster proximity that differentiates the various agglomerative hierarchical techniques that we will discuss. Cluster proximity is typically defined with a particular type of cluster in mind. For example, many agglomerative hierarchical clustering techniques, such as MIN, MAX, and Group Average, come from a graph-based view of clusters. MIN defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph

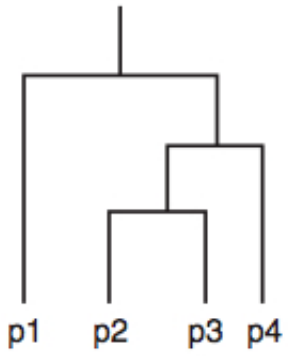


Figure 1.7: example of a dendogram

Algorithm 8.3 Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary.
2: **repeat**
3: Merge the closest two clusters.
4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
5: **until** Only one cluster remains.

Figure 1.8: Basic agglomerative hierarchical clustering algorithm

terms, the shortest edge between two nodes in different subsets of nodes. Alternatively, MAX takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes. (If our proximities are distances, then the names, MIN and MAX, are short and suggestive. For similarities, however, where higher values indicate closer points, the names seem reversed. For that reason, we usually prefer to use the alternative names, single link and complete link, respectively). Another graph-based approach, the group average technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters. Figure 1.4.2 illustrates these three approaches.

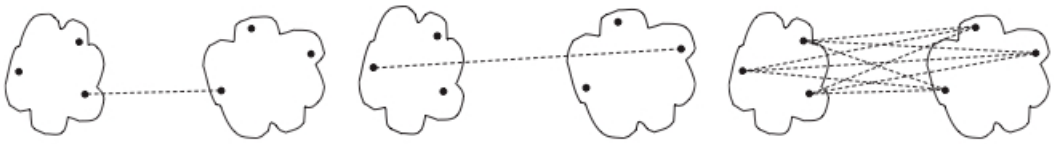


Figure 1.9: Graph-based definitions of cluster proximity

If, instead, we take a prototype-based view, in which each cluster is represented by a centroid, different definitions of cluster proximity are more natural. When using centroids, the cluster proximity is commonly defined as the proximity between cluster centroids. An alternative technique, Ward's method, also assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the SSE that results from merging the two clusters. Like K-means, Ward's method attempts to minimize the sum of the squared distances of points from their cluster centroids.

The Calinski and Harabasz Stopping Rule

In [1] a method for identifying clusters of points in a multidimensional Euclidean space is presented. An informal indicator of the best number of clusters k is also calculated. The method supposes there are n individuals with values of the same

v variables for each individual. These individuals can be represented by n points in a v-dimensional Euclidean space. An $n * n$ distance matrix is then calculated. Next, the method needs to calculate the Minimum Spanning Tree (MST), so that the enormous number of possible partitions of a set of points is reduced to those which are obtainable by splitting the MST.

This tree is then partitioned by removing some of its edges. If we want to divide the n points into k clusters, $k-1$ edges have to be removed. For each possible partition, the within-cluster sum of squared distances about the centroids is computed. In order to calculate the optimal value of k, first $k = 2$ is taken, then $k = 3$, and so on. For each value of k, the best partition is calculated with the minimum W GS S and the Variance Ratio Criterion (V RC):

$$VRC = \frac{\frac{BGSS}{K-1}}{\frac{WGSS}{n-k}} \tag{1.18}$$

where BGS S is the total between-cluster sum of squared distances. The authors suggest using V RC as an informal indicator for the best value of k. They also suggest the computation of V RC for $k = 2, 3, ...$ choosing the value on k for which the VRC has an absolute or local maximum. The computation can be stopped when the first local maximum is reached. Although of working with the minimum spanning tree instead of the whole graph reduces the number of partitions to be examined, this number,

$$\binom{n-1}{k-1} \tag{1.19}$$

is high enough to use this method with even moderate value of n.

Chapter 2

DATA PREPARATION

2.1 Data sets description

Before starting with data manipulation and transformation we give a brief description of the database we have to work with. In figure 2.1 we give a conceptual diagram according to Calinsk notation for geo-referenced databases.

In next subsections we will describe in detail each entity in showed figure.

2.1.1 Data set of streets

The data set of streets has a crucial role in our project, it is the starting point of many following tasks. In fact its manipulation could be intended as the core of new ideas presented with this thesis.

Cermit_Knoten_Gem_05 is a street network, a digitalized map of all streets in Germany. The smallest units are street segments, which usually denote the part of a street between two intersections. Each unit is identified by a geometric shape (line) which belongs to a topologically correct Geographic Information System. Each row in the data set describes a Navteq street segment with a geometry (line) and attributes.

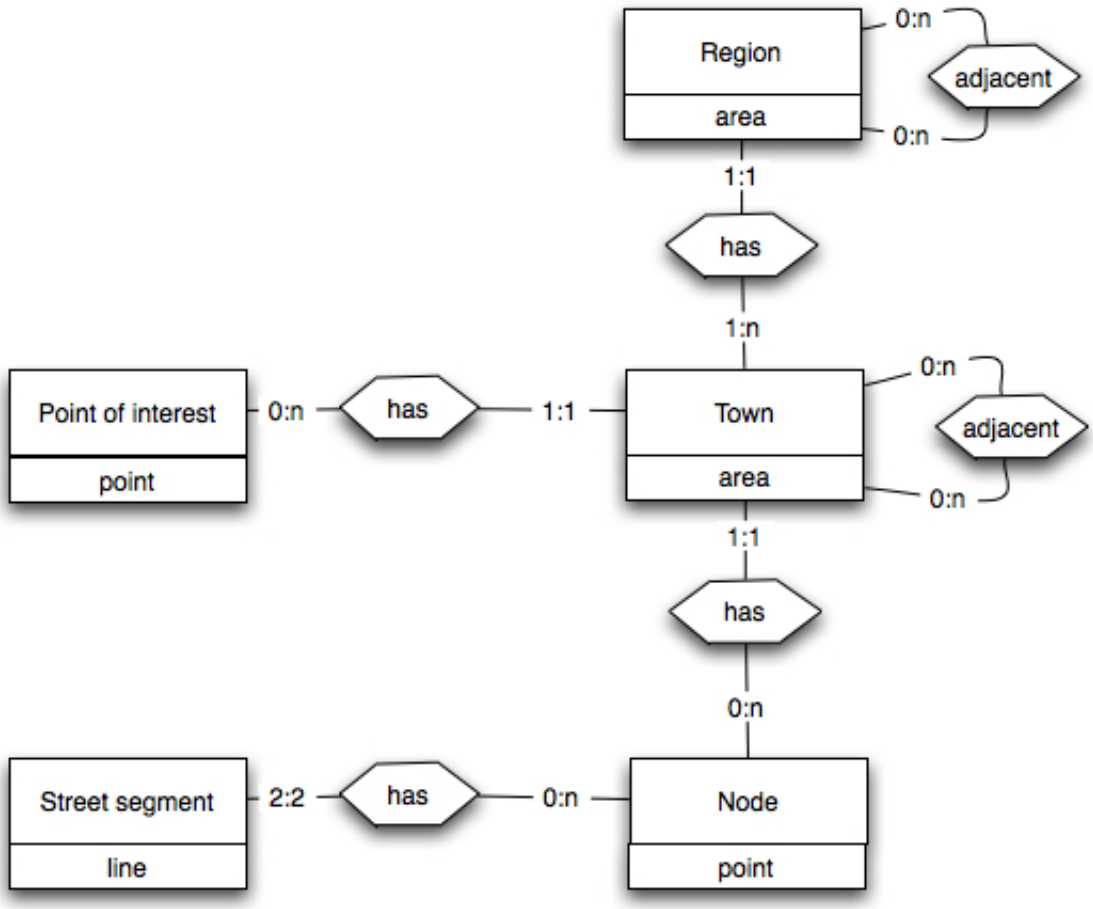


Figure 2.1: The E-R Diagram for the geo-referenced database

The data set is provided by DDS (Digital Data Services GmbH) and consists of 5475436 segments and 4390984 intersections. In the following we trace in detail the interesting attributes in the format *Attribute_name*[data_type]: description:

- *Prim_name*[Char (40)]: official denomination of the street;
- *Sek_name*[Char (40)]: alternative name of the street;
- *Type*[Short Integer]: the type stays for the driving speed permitted in that street segment. type 1-3: highway (Autobahn) fast/medium/slow - type 4-6: federal road (Bundesstrae) fast/medium/slow - type 7-9: country road (Landstrae) fast/medium/slow - type 10-13 city road (Stadtstrae) fast/medium/slow - type 13: ferry - type 14: slow inner city road - type 15: special cases (Zone 30, pedestrian area, forest road);
- *Kat*[Short Integer]: the category gives the meaning of importance to the street. The smaller the number, the more important the road. kat 1: main roads (highways) - kat 2: first class roads - kat 4: second class roads - kat 5: third class roads - kat 7: fourth class roads (side road);
- *Von*[Integer]: starting point of the street segment;
- *Nach*[Integer]: ending point of the street segment;

2.1.2 Data set of towns

The whole Germany is divided into 12503 territorial entities representing towns, in our data set uniquely identified in the table *Gemeindeauswahl* by a GKZ (postal code). Each territorial entity is described by a geometrical shape (area). The data provided by DDS (Digital Data Services GmbH) have a certain quality, they are topologically correct. This means there are no gaps between two adjacent areas, geographical primitives of town territorial entities.

The SCOT's goal concern to find groups in a subset of towns. Therefore the object of our work will not be the whole data set, but a subset which is easier to call *focus towns*. The *focus towns* are 2482, figure 2.1.2, mainly the towns between 10 and 50 thousands inhabitants plus a number of towns with less than 10 thousands. Anyway the whole data set of towns will be useful for our purposes.

2.1.3 Data set of nodes

The table *Cermit_Knoten_Gem_05* maps the town membership for the starting or ending point of street segments. If a point belongs to two towns, it means that it lies on a boundary line, there are two rows associating the same point to two different towns.

2.1.4 Data set of points of interest

The data set includes all points of interest in Germany. The table has a unique identifier(*ID*) of the point of interest, the name(*Name*), the type(*Typ*) and the geometry(*Geometry*), which is normally a point, marking the location in space. The table of points of interest contains circa 150.000 observations provided by DDS (Digital Data Services GmbH). The table 2.1 has a point of interest called *(*Central Station) identified by a type and a point in a bi-dimensionl space $x = 12,5$ and $y = 58$.

ID	Name	Typ	Geometry
1	Central Station	10	(12,5 ; 58)

Table 2.1: Table - Point of interest

2.2 Spatial object representation

Two-dimensional object representation and recognition by computer began in the 1960s and has been an active area of research ever since. The method used for

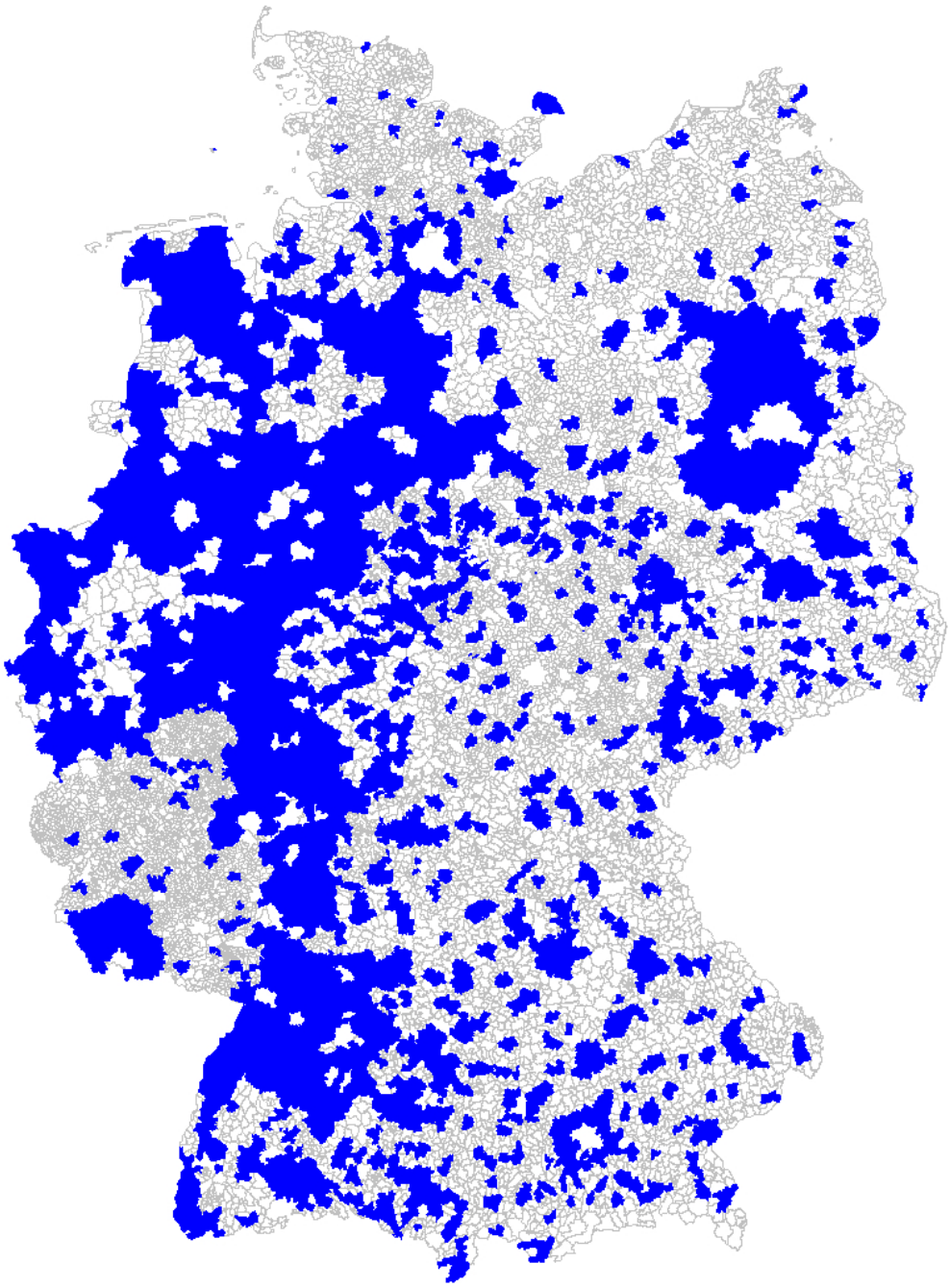


Figure 2.2: The *focus towns*

recognition often depends on the particular representation selected. The topic covers a deep variety of object representation: representation by global features, by local features, by boundary description, by skeleton. We focus on the one wich best fits the problem faced: representation by local features.

A two-dimensional object can be characterized by its local features, their attributes, and their interrelationship. The most commonly used local features are corners that imply a certain connection. Local features must be organized into some type of structures for matching. The most common type of structure is a graph whose nodes represent local features and their properties or measurements and whose edges represent relationship among the features 1.2.

In the following two sections we will describe the two representations used to infer on spatial objects. The first subsection describes the fundamental data set used in this section (data set of streets), the second shows the representation of global relationship between towns and the third the representation of inner street network.

2.2.1 Global representation between towns

The global representation between towns intends to give a basic structure, representing the street connections between all towns in Germany, which is useful to infer new information on spatial objects (German towns), in order to find some similarity between them with respect to their traffic frequencies. In fact we will exploit such representation as the starting point of section 2.3 and 2.4. In the next three sections we will describe how we have found the connecting towns, how we defined distance as the main property of relationships, and how we stored the computed data in an adjacency matrix representing a graph.

Connecting towns

Given the previous data, 2.1.1, there are three cases (figure 2.2.1) in which two towns are connected and handling them we are able to find all relationship between towns.

To facilitate the understanding we assume that the starting point of a segment is always the nearest to the centroid of a town area.

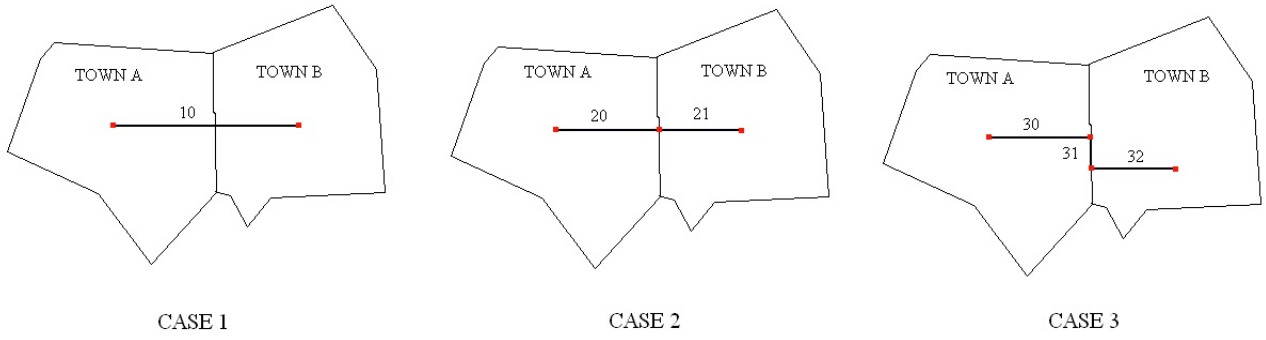


Figure 2.3: The three cases of connected towns

The first case catches all connections in which there is a segment having a starting point belonging to one town and an ending point belonging to another. In the figure 2.2.1 *case 1* the segment 10 has the starting point in $TOWNA$ and the ending point in $TOWNB$. This case is solved with a *SQL* query and the resulting output pertain to 93.448 connections. The rows of output table are in the form *TOWN A*, *TOWN B*, segment 10.

In *CASE 2* of figure 2.2.1 there are two segments collapsing in a boundary line. In the figure 2.2.1 *CASE 2* the segment 20 has the starting point in $TOWNA$, the ending point on the boundary line, the segment 21 has the starting point in $TOWNB$, the ending point on the boundary line. The complexity grows, but we are still able to solve the problem with a *SQL* query. The output table contains 8699 connections, and it is in the form *TOWN A*, *TOWN B*, segment 20, segment 21.

The third case include all remaining connections between towns. In this situation we have a segment with starting and ending point belonging to both towns, so lying

on a boundary line. In the figure 2.2.1 *CASE 3* the segment 31 has the starting and ending point on a boundary line and the segments 30, 32 converge into them. A *SQL* query could not handle the problem, therefore we opted for *R-Project* statistical tool that offers a wide range of tools to manage database in an easy way. The results table shows 1876 connections, obtained aproximately within a time of approximately 60 hours.

The algorithm creates a list for each boundary street. We remember that a street may also consist of many segments. If there are two street segments with starting point belonging to two different towns and their ending point belong to the same list of boundary segments, it is recognized as connection. In figure 2.2.1 *CASE 3* the list of boundary segments consists only of segment 31. Then the algorithm checks if there are segments with starting point belonging to different towns and having the ending point equal to a point in the same list. They are segments 30 and 32 that have their ending points equal to two points of 31 and they belong respectively to *TOWN A* and *TOWN B*. The connection will be traced by: *TOWN A*, *TOWN B*, segment 30, segment 32.

The aim of discovering connections was only to know which towns are connected, so it was not necessary to preserve information on segments. In any event we considered that this information may be useful for future work. Anyway the result of this task is a database table in which we know all the connecting adjacent towns for a certain town.

Defining distance

At this point all the connections between towns are available. The discovery of a relationship between two towns is interesting, but a more accurate information could be more helpful. Therefore we defined a distance property for each relationship, based on the Kilometer unit. Oracle has a set of tools available to handle spatial information : Oracle Spatial. In particular we used the function $SDO\_GEOM.SDO\_DISTANCE$ $(d.geo\_1, d.geo\_2, 0.005, 'unit = km')$ that computes the distance between any two

objects (points, lines, areas), in this case the objects *d.geo_1* and *d.geo_2*, in respect of their reference system.

After computation the resulting output will be in the form $TOWNA$, $TOWNB$, $DISTANCE$. It represents all the information we need to start inferring service level, section 2.3, and paths, section 2.4, but its data structure does not fit the computation we are going to apply efficiently. In the following we describe the matrix data structure of the graph, a useful way to represent the relationships.

Graph representation

The relationships found could be stored in a graph. We know there are many ways to represent a graph. One is the adjacency list, which is more efficient in memory storage. However an adjacency matrix data structure is more suitable for our purposes because we will need a memory usage of $n * n$ (n as number of vertices) to store the distance from all possible town connections, section 2.3, and secondly because it is a trade-off in order to optimize the two algorithms, one is the last mentioned and the other is the discovering path algorithm in section 2.4.

So the graph is defined by $G(V, E)$, where nodes V are towns and edges E are relationships between towns. The $distance(u, v)$, where $u, v \in V$, is the property of every edge describing the distance in kilometers of the connection. Now we explain how we stored the graph G in an adjacency matrix data structure. The number of vertices V and edges E in G amount respectively to 12503 and 48339.

The graph G is stored in a matrix that contains in first column and first row the list of vertices in V in G. For each relationship $TOWNA, TOWNB, DISTANCE$ we look for the cell location of $TOWNA$ in the first column and the location of $TOWNB$ in the first row. In the corresponding cell, localized by the couple of coordinates found, we store the respective distance. If there is no connection between two towns we store the value 0. The resulting matrix has in its diagonal a list of zeros because a vertex has no connections with itself, and it is an upper triangular matrix. The storage space is relevant: $596.4Mb$.

2.2.2 Inner street network representation

As in the previous spatial representation, the medium to infer new information is a graph. It intends to represent street networks of a georeferenced database, in a way in which is possible to infer useful and more general information. First only main streets are selected, then a new topology is defined and in the third subsection they are stored in a graph data structure.

Selecting main streets

Even relatively small have big amounts of street segments that could disturb the discovering of street network structures. For this reason we decided to choose a higher level of detail and select only main streets. The importance of a street segment is explicated by its category, the smaller the category, the more important the street. Accordingly the streets segments are selected with categories from 1 to 4 and as a consequence the ones from 5 to 7 are removed. In figure 2.2.2 left a town with all its street segments is presented and in figure 2.2.2 right the same town selecting only streets with category 4 or less.

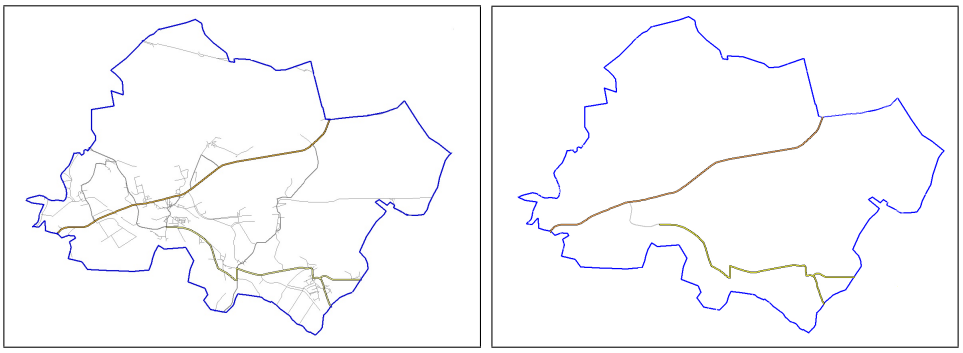


Figure 2.4: Example selecting main streets

Defining a new topology

The street selection brings to light the need of a new topology. First we analyse the figure 2.2.2 left where is possible to see the topology with all streets segments, with categories from 1 to 7. If a street has a shared point with another street, it is divided into more segments. In the example a street is divided into segments 4 and 5 because of a shared point with segment 3. In fact even if the segments 4 and 5 belong to a unique street, we have two distinct entities. The selection implies avoiding all segments with category higher than 4 as in the example of figure 2.2.2 right. The red segment 3 is category 5, so in the new topology it is not represented. Therefore the street in the old topology formed by segments 4 and 5 actually has no further shared points with segment 3, and it is represented in the new topology with only the segment 4.

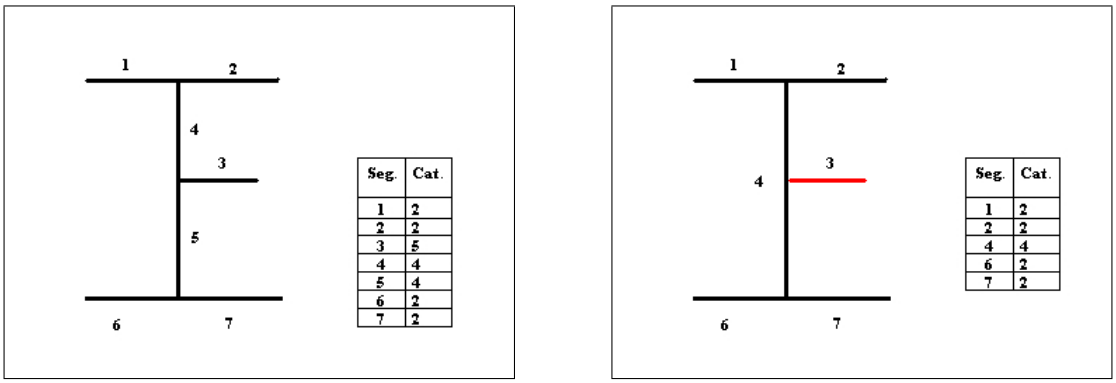


Figure 2.5: Example new topology

Graph representation

The remaining geo-referenced segments need to be represented in a way in which the inference of more general information is possible. Every town is computed as a Graph $G := (V, E)$ where $u \in V$ is a set of vertices representing street segment, and E is a set of edges $E = u, v|u, v \in V$ that represent two connecting segments. Each vertex

$u \in V$ has its category given by the function $category(u)$. In figure 2.2.2 we give an example. The town has two streets intersecting in the red point, therefore there are four street segments: b , c , d , e.

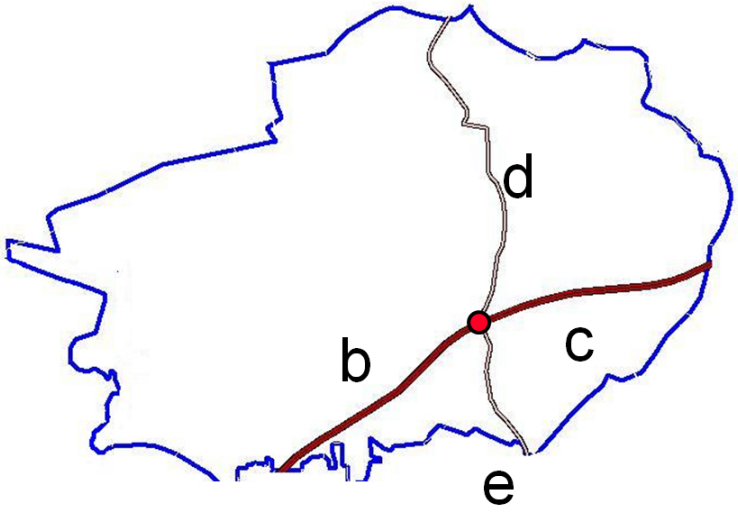


Figure 2.6: Street network graph extraction example

The street segments become vertices: $V = b, c, d, e$, figure 2.2.2. The street segment identified by b has a shared point with segments c, d and e, therefore in the graph there will be three edges: (b, e), (b, c) and (b, d). The same reasoning is used for segments c, e and d discovering respectively (c, e),(c, d) for c and (e, d) for e.

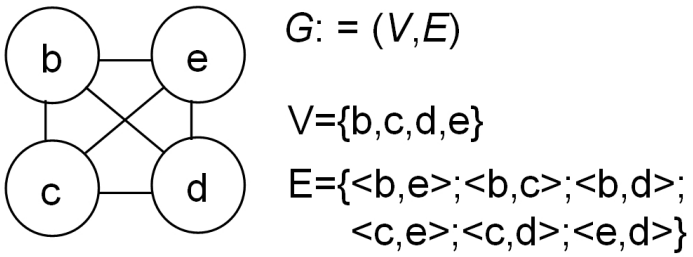


Figure 2.7: Street network graph extraction example 2

Could be more intuitive to define the street segments as edges and the intersections as nodes of the graph, but this representations do not fit the matching algorithm for relational distance 1.3.3 we are going to apply in the section 2.5.1.

2.3 Service Level

The chapter inteds to describe in detail service level task, one of three main tasks of spatial data preparation. Service level could be intended as an index denoting how a town is served by the neighborhood towns. For example in terms goods, administrative or political issues or just entertainment, in other words all can make the people move from a town to another.

The *Central Place Theory* 1.1.2 is a geographical theory that seeks to explain the size and spacing of human settlements. It rests on the notion that centralization is a natural principle of order and that human settlements follow it [19]. It gives us the basic notions on how to extract the Service Level from data, we explain this task in the section 2.3.1.

The goal of task is to group together towns with the same service level. If fact two towns with the same service level probably have some towns around which offer equal kind of services. This means that the inhabitants in both towns could drive with same frequencies to neighborhood towns. So after tracing the meaning of service level in next section, the 2.3.2 groups together towns with similar service level.

2.3.1 The algorithm for discovering service level

We obtained a graph $G(V, E)$, where nodes V are towns and edges E are streets connection between two towns, by computing all the streets in Germany . The graph is the starting point and the fundamental tool of the algorithm. In its matrix form whether the connection is direct it has a connection, an edge (u, v). In other words the street that connects the towns u and v does not cross another town z. The following will present the two steps of the algorithm, the first is the computation of a graph including indirect street connections and the second is the application of Central Place Theory conditions.

Obtaining a Graph with indirect street connections

We were looking for the shortest paths so the Dijkstra algorithm was the obvious choice of method to use as it harmonizes simplicity and efficiency 1.3.2. It is a greedy algorithm that solves the single-source shortest path problem for a directed graph with nonnegative edge weights. The input of the algorithm consists of the weighted directed graph G. Weights of edges are given by a weight function $w : E \to [0, \infty)$; distance in Kilometers. The distance of a path between two vertices is the sum of distances of the edges in that path. For a given pair of vertices s and t in V, the algorithm finds the path from s to t with shortest distance. It computes shortest paths from the vertex u of V to all other $V \backslash \{u\}$ vertices in the graph, and then the same for all remaining $V \backslash \{u\}$ vertices. In the resulting output each town is connected to all remaining towns with the shortest path.

Applying Central Place Theory conditions

In this step the minimum population and the radius examined in Central Places Theory play a crucial role. According to the conditions of theory 1.1.2, every town u in V has a category (from *MARKTORT* to *LANDSTADT*) defined by its population, and that implies a proper radius, $radius(u)$. Then the amount of vertices x in V with an edge (x, u) which satisfy the condition $distance(x, u) < radius(x)$ are counted for each vertex u in V. The output of this phase is a table in which columns from one to seven contain the number of towns, from Marktort to Landstadt, covering the town identified in the corresponding row, an example of which is shown in figure 2.8. The columns define the service level for each city in the row. Explaining the example, *TOWN 1* is within the radius of two towns classified as *MAKTORT*, of four towns classified as *AMTSORT* and of five towns classified as *LANDSTADT*.

Computational time

The algorithm is run on a computer with an Intel® Core™2 Duo Processor and with 2 GB Ram DDR. While most of time it is used for indirect connection computation,

	MARKTORT	AMTSORT	...	LANDSTADT
TOWN 1	2	4	...	5
TOWN 2	0	8	...	3
...	...	...	...	...

Figure 2.8: Table example output applying Central Place Theory coditions

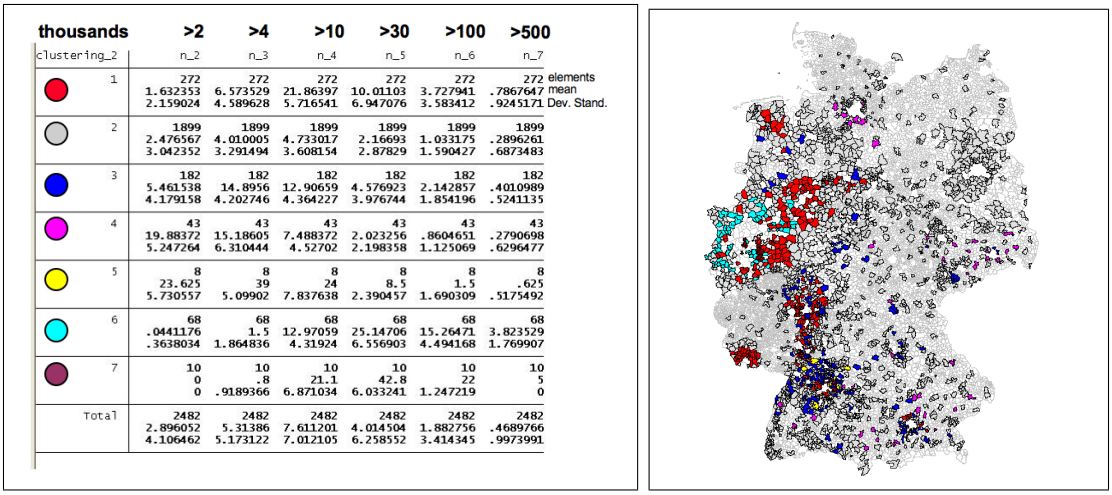
this algorith in fact, Dijkstra 1.3.2, has a $O(e * \lg(e))$ complexity, where e is the number of edges in the graph. So the complexity depends on the number of edges, 48339, that bring about a total computational time of circa 20 hours.

2.3.2 Service level clustering

In this section the output of previous algorithm is given as input to a hierachical clustering algorithm to discover any similarity between towns according to their service level. The average linkage algorithm 1.4.2 suits our problem well. A useful method to acquire knowledge about the best number of clusters is the Calinski and Harabasz method 1.4.2. The resulting number is seven, so the dendogram obtained by the average linkage algorithm is cut where the groups are equal to such number. In figure 2.3.2 left we can discern the number of elements, the mean and deviation standard of variables in each cluster; in figure 2.3.2 right we see the distribution of towns in a thematic map of Germany corresponding to cluster affiliation.

Evaluation

On closer inspection of figure 2.3.2 left we realize that cluster seven is strongly served especially by big towns. Its mean values for *Landstadt* (5.0), *Provinzstadt* (22.0), *Gaustadt* (42.8) and *Bezirkstadt* (21.1) are higher than total mean, respectively (0.4), (1.8), (1.0) and (7.6). Cluster six is served over total mean by big towns, but in a less extensive way; we could call it 'cluster substantially served by big towns'. In fact its mean values are *Landstadt* (3.8), *Provinzstadt* (15.2) and *Gaustadt* (8.5) are higher than the total mean, respectively (0.4), (1.8), (1.0) and (7.6). This could mean that



thousands	>2	>4	>10	>30	>100	>500	
clustering_2	n_2	n_3	n_4	n_5	n_6	n_7	
1	272	272	272	272	272	272	elements
	1.632353	6.573529	21.86397	10.01103	3.727941	.7867647	mean
	2.159024	4.589628	5.716541	6.947076	3.583412	.9245171	Dev. Stand.
2	1899	1899	1899	1899	1899	1899	
	2.476567	4.010005	4.733017	2.16693	1.033175	.2896261	
	3.042352	3.291494	3.608154	2.87829	1.590427	.6873483	
3	182	182	182	182	182	182	
	5.461538	14.8956	12.90659	4.576923	2.142857	.4010989	
	4.179158	4.202746	4.364227	3.976744	1.854196	.5241135	
4	43	43	43	43	43	43	
	19.88372	15.18605	7.488372	2.023256	.8604651	.2790698	
	5.247264	6.310444	4.52702	2.198358	1.125069	.6296477	
5	8	8	8	8	8	8	
	23.625	39	24	8.5	1.5	.625	
	5.730557	5.09902	7.837638	2.390457	1.690309	.5175492	
6	68	68	68	68	68	68	
	.0441176	1.5	12.97059	25.14706	15.26471	3.823529	
	.3638034	1.864836	4.31924	6.556903	4.494168	1.769907	
7	10	10	10	10	10	10	
	0	.8	21.1	42.8	22	5	
	0	.9189366	6.871034	6.033241	1.247219	0	
Total	2482	2482	2482	2482	2482	2482	
	2.896052	5.31386	7.611201	4.014504	1.882756	.4689766	
	4.106462	5.173122	7.012105	6.258552	3.414345	.9973991	

Figure 2.9: Service level clustering analysis

clusters seven and six are quite close from big towns and depend on them a great deal for service. The colors turquoise and reddish purple in figure 2.3.2 right correspond to towns belonging to cluster six and seven. In fact this area is called "Ruhrgebiet". It is located in western Germany and it is a densely populated industrial area. The industry grew historically from mining and steel production during the industrial Revolution and now districts have grown into a large complex forming an industrial landscape of unique size, inhabited by some 5.3 million people, the fifth largest urban area in Europe.

Clusters five and four have higher mean values for emphAmstort being (23.6) and (19.8) respectively, and emphKreissstadt (39.0), (15.1) than total mean, emphAmstort (2.8) and emphKreissstadt (5.3). We could call cluster five 'independent covered by medium-small towns' because the towns in this group are quite far from big ones; cluster four 'independent isolated covered by medium-small towns' because of its lower mean values than cluster five.

The remaining three clusters accord generally with the total mean, with some variation for cluster one that seems to have towns more served by big towns, *Provinzstadt* (3.7) and *Gaustadt* (10.3), and for cluster three that seems to have towns served by medium-small ones, emphKreissstadt (14.8) and emphBezirkstadt (12.9).These two

clusters, the blue and red towns in figure 2.3.2 right, below follow the river Rhein, which also indicates populated area. Water for ancients was synonym of life and many big towns with old origin are located along the Rhein and its main tributary Main: Kln, Bonn, Koblenz, Frankfurt, Wiebaden, Mainz, Mannheim, Ludwigshafen, Karlsruhe. Cluster two is the biggest with 1899 towns and for this reason it is more difficult to characterize it.

2.4 Discovering paths

The second main task of spatial data preparation is the discovering of paths and it will be presented in detail in this chapter.

At the beginning the idea was to discover all frequent patterns of town relationships. Relationships could be for example a small town on a river or in a mountain, and therefore to make them explicit adding new dimensions to our final data set. It could be really interesting, but we had to optimize the trade-off between time (operative and computational) and quality of results. For this reason we decided to carry on considering only the patterns of paths we are going to describe, we think they can give the best results for our problem.

The global relationships between towns is relevant, the capturing long distance traffic and paths of commuters is the implicit aim of this model. If we can discover long paths we can determine which towns belong to the paths. So we are able to discover which towns are in path between two other towns. In particular we consider only the *focus towns* between two bigger (in term of number of inhabitants) towns.

We are supposing that *focus towns* could have some common characteristics where they lie in a path defined by main streets between two bigger towns. It easy to imagine that the frequency of Federal highway that passes a focus town between two bigger towns might be higher due to commuters than a Federal highway in a *focus town* of comparable size that lies by itself.

That is why the goal of this task is marking such *focus towns*. The output is a flag field for each town denoting if the town is a town in a path or not, so 1 or 0. In the next sections we describe the algorithm and its evaluation.

2.4.1 Algorithm

The graph obtained $G(V, E)$ in 2.3.1, where nodes V are towns and edges E are streets connection between two towns, is filtered by including only streets that are motorways (Autobahn) or federal highways (Bundesstrasse). There is an edge between vertices

u and v of V only if the edge (u, v) is extracted by a street with type 1 or 2. The next step computes a Depht-first search (DFS, 1.3.2) with deep limit 15 for every vertex u on Graph V where the function $inhabitants(u)$, giving the number of inhabitants of u, is bigger than 50 thousands. This means it is considering for starting point of a DFS only towns bigger than focus towns. In fact focus towns have a number of inhabitants smaller than 50 thousands.

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one begins at the starting node and explores as far as possible along each branch before backtracking. Beginning from vertex u each time it explores another vertex x where $inhabitants(x) \geq 50.000$ it stores the path from u to x. The shortest path for that couple u and x is examined: if there are one or more focus towns in it the algorithm marks them as focus towns in a path. The resulting output of the algorithm is a value, one or zero, for each focus town denoting if it lies in a path or not. An example is given in the figure 2.10; the green towns have more than 50.000, the blues and reds are focus towns, not in a path and in a path respectively. It means that in the shortest path between green towns there is a focus town, the one marked in red.

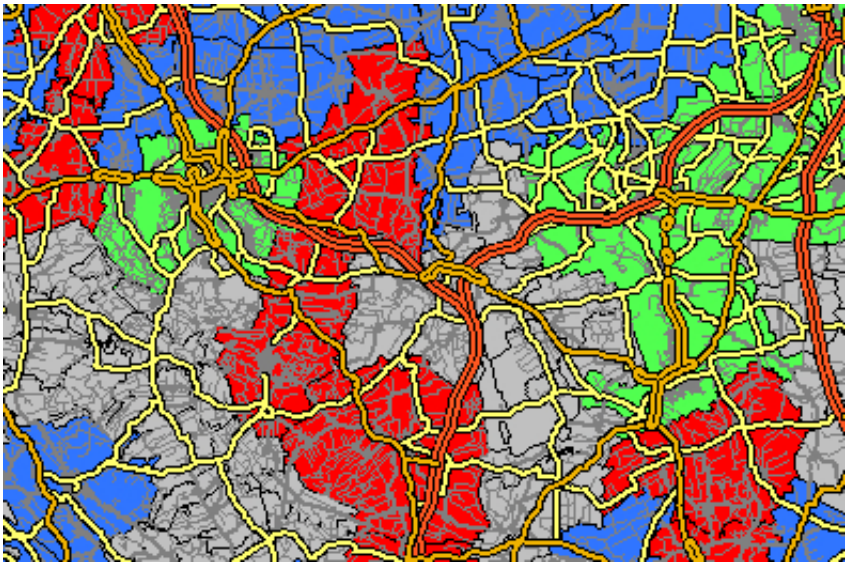


Figure 2.10: Example of focus towns in a shortest path

computational time

The amount of data is important, the total number of nodes is 12.503, of which 192 have more than 50.000 thousands inhabitants, and the total number of edges is circa 9.000. For each of 192 towns a DSF is computed and the total time needed is about 48 hours in a Intel® Core™2 Duo Processor with 2 GB Ram DDR.

2.4.2 Evaluation

The algorithm discovers 359 focus town in a path, the red towns in figure 2.11. It captured the western north-south connection along the river Rhein, and also found many *focus towns* in the densely populated Ruhrgebiet. There are traces of east-west connection from Ruhrgebiet over Hannover to Berlin. But connections in eastern Germany are rare. It captured also *focus towns* between Dresden, Chemnitz, Zwickau and Plauen, which are alongside the mountain Erzgebirge. On closer examination we could say that there are better results when the depth of DFS is shorter, anyway even longer, until depth 15 we find the results a good approximation.

If the algorithm finds two paths with the same cost, it marks *focus towns* in a path all towns in both paths. This is our choice, however in future works the algorithm could be improved in order that the paths are chosen analyzing other variables which discriminate the best one. For example the best path between two towns could be the quicker in term of time.

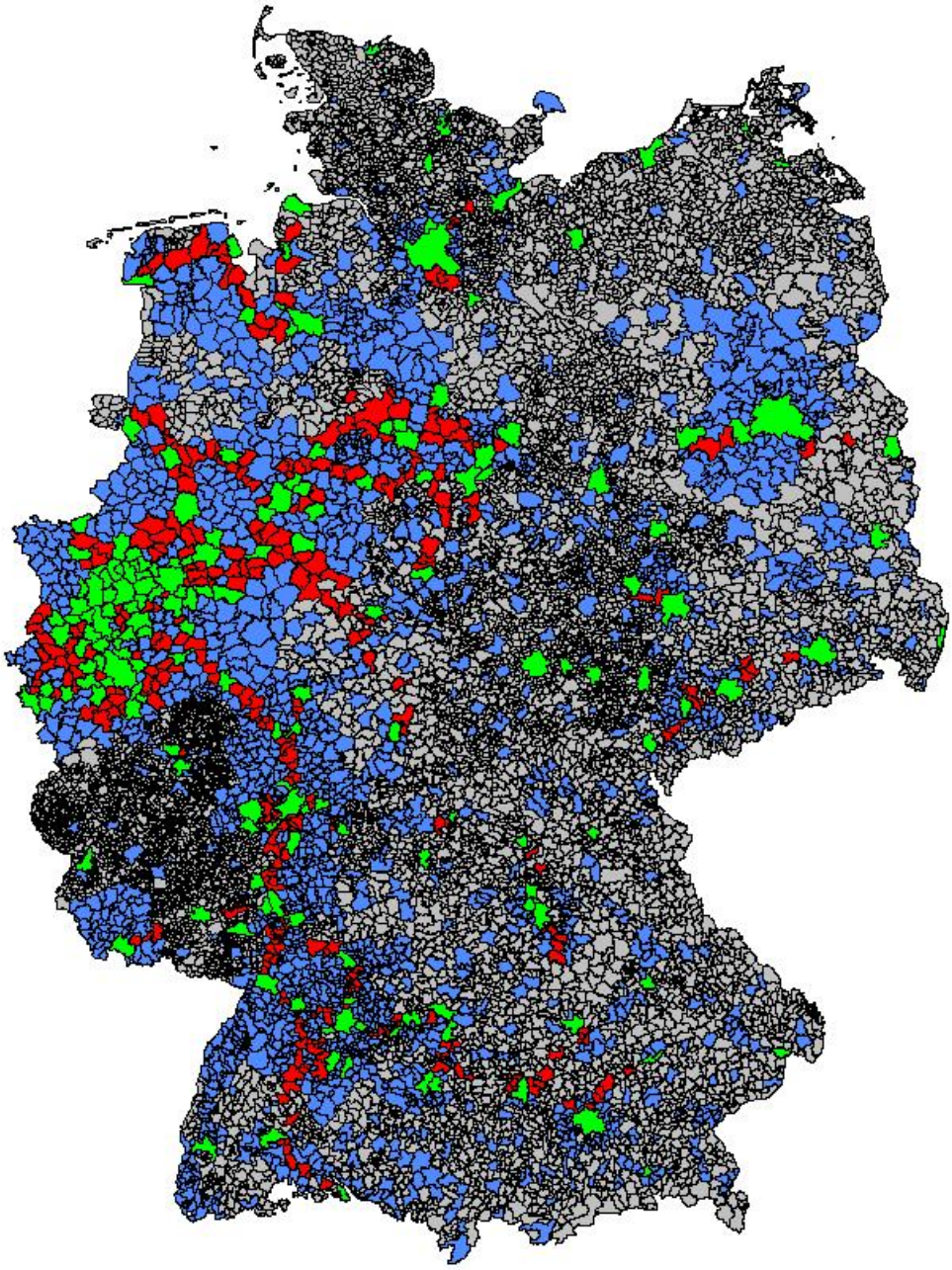


Figure 2.11: Output of discovering paths algorithm in a thematic layer

2.5 Street network structures model and matching

Urban Geography provides a plentiful literature about the street networks of German towns. As presented in 1.1 there are different kinds of street networks. The idea is to examine this evidence to find another dimension of similarity for focus towns. The final goal is to find clusters of towns with street network similarities, and it is not difficult to realize that there is dissimilarity between the two towns in figure 2.5.

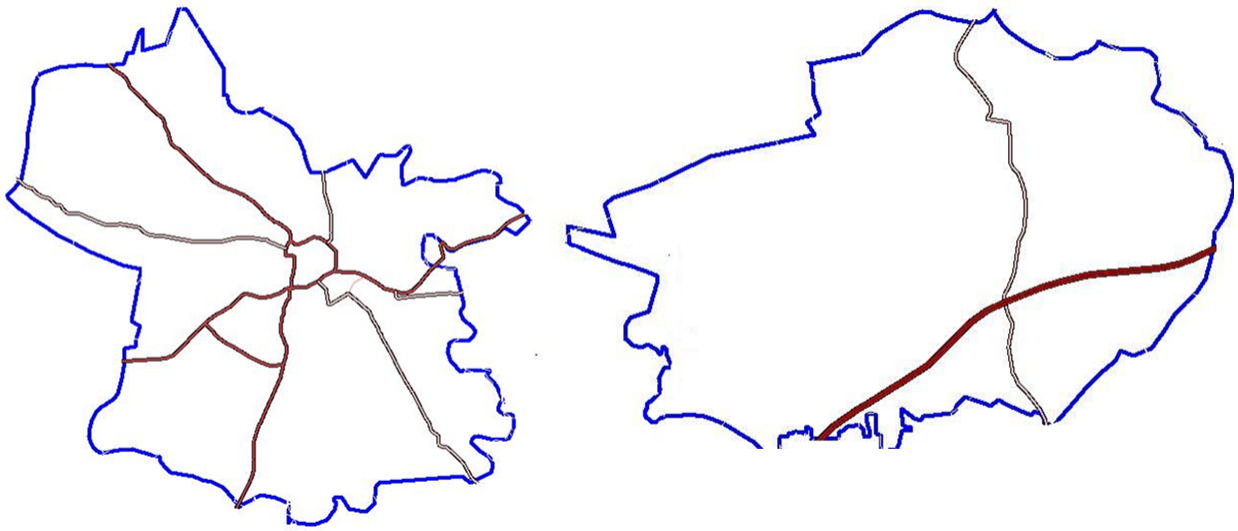


Figure 2.12: Street network structure examples

Object recognition plays an important role in this chapter. In order to recognize and identify objects, we must have one or more models of the object, in our case the street network model, that may appear in the universe it deals with, and we do not have such models. It is an unsupervised task, which is why we are going to identify some models through a clustering of the street network. Section 2.2.2 discusses how to represent a street network. Here we describe the matching process and the discovery of street network models through a clustering.

2.5.1 Matching and Clustering Street network representations

Presently the goal is to compare graphs and find groups with some similarity. The concept of Matching in section 1.3.3 is fundamental here. Matching means finding a correspondence between two entities, and it can be done by a relational distance 1.3.3 that can compare two structures determining their relational similarity.

The result of section 2.2.2 is a number of 2482 graphs concerning focus towns. At this point the vertices of the graph are the intersecting point between streets, and the edges are streets with category 4 or less. It is not an intuitive solution, someone could say that could be better to switch the representation of streets in nodes and of intersecting points in edges. But we do not think so, this representation fits the relational distance 1.3.3 which needs an attribute for edges, in our case the category of the street.

Matching with Relational distance

At this step every graph is compare with each other. We will use as an example the graphs q and d in Figure 2.5.1, the first having six edges end the second two. We know that the nodes $1, 2, 3, 4 \in q$ and $a, b, c \in d$ have properties defined by function $p(node)$; respectively $p(1) = x,\ p(2) = y,\ p(1) = z,\ p(1) = x$ and $p(a) = x,\ p(b) = y,\ p(c) = w$. The edges for q and p are respectively $< 1, 2 >;< 1, 3 >;< 1, 4 >;< 2, 3 >;< 2, 4 >;< 3, 4 >$ and $< a, b >;< a, c >$. A possible mapping f from A to B is $f(1) = a;\ f(2) = b$. For this mapping we have

$$
\begin{aligned}
|(R - (S \circ f^{-1})| &= |\{< 1,2 >;< 1,3 >;< 1,4 >;< 2,3 >;< 2,4 >;< 3,4 > + \\
&\quad - (< a,b >;< a,c > \circ f^{-1})\}| \\
&= |\{< 1,2 >;< 1,3 >;< 1,4 >;< 2,3 >;< 2,4 >;< 3,4 > - < 1,2 >\}| \\
&= |\{< 1,3 >;< 1,4 >;< 2,3 >;< 2,4 >;< 3,4 >\}| = 5
\end{aligned}
$$

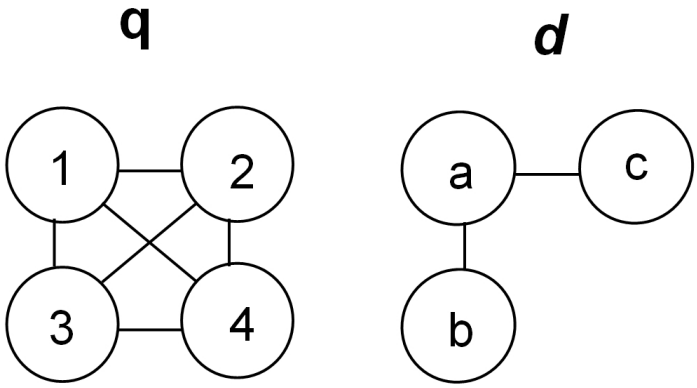


Figure 2.13: Graph q and d for example of relational distance

$$
\begin{aligned}
|(S-(R\circ f)| &= |\{<a,b>;<a,c>+ \\
&\quad -(<1,2>;<1,3>;<1,4>;<2,3>;<2,4>;<3,4>\circ f)\}| \\
&= |\{<a,b>;<a,c>-<a,b>\}| \\
&= |\{<a,c>\}| = 1
\end{aligned}
$$

$$
\begin{aligned}
GD(q,d) &= \frac{I-|(R-(S\circ f^{-1})|}{I} + \frac{L-|(S-(R\circ f)|}{L} \\
&= \frac{6-5}{6} + \frac{2-1}{2} = 0.666
\end{aligned}
$$

The perfect matching results in a score of $GD(q,d) = 2$. In this case q and d have the same number $x = y$ of edges, the x edges in d match the y edges in q and vice-versa. The maximum dissimilarity results in $GD(q,d) = 0$, when no one edge of q matches edges in d and vice-versa.

The computational time is about 8 hours and the resulting output is a similarity matrix $n * n$, where $n = 2482$ is the number of focus towns.

Street network models

Many attemps with different algorithms are made. The tight range of discance values (0-2) and the big number of observations do not help to find a small number of groups. Hierarchical algorithms like average linkage and single linkage 1.4.2 could not compute reasonable results. The final number of groups with such algorithms is not less than 500, a number too big if we think that it brings about a mean of five observations per group in a dataset of 2482 observations. Single linkage has better results because of its kind of reasoning. This clustering results in a number of 312 groups, a better result, but not exciting.

The reduction of groups is an hard task with the similarity matrix of graphs. For this reason we worked to a less sophisticated way to threat street network information. This alternative solution consists of ignoring the adjacency between street segments and to compute only the amount of segments for each category. Each of 2482 rows will have four columns, one for each street category, containing the amount of street segment of a certain category. With such dataset the final number of groups is reasonable. We find it just maximising the similarity within groups and minimizing the similarity between groups with Calinski-Harabaz method 1.4.2. The best number results in 12.

Really we do not know at the moment which of two solutions presents better results because the models have to investigated with future works. Paradoxically could be that the more sophisticated solution has worse results.

2.6 Non-spatial data preparation

The synthesis of the geographical information has to be merged with available non spatial information. But first the non-spatial information needs to be manipulated.

The Data mining technologies have the objective to wring use and meaning out of data. Technologies themselves are not an answer, they are tools to help find an answer. Data has equally importance, It has to be readied to find the best answer to the question asked. In the following two sections we tackle this problem both for tourist information and for further town information.

2.6.1 Tourist information

The tourist information proves interesting. It could add a meaningful dimension to our domain. In other words where two towns have the same tourist information they will be closer in the n-dimensional space and so more similar. Same tourist information in our case means to have the same number of tourist attractions. For example a town with seven tourist attractions will be more similar to a town having five tourist attractions than another having three. The number of points of interest is 150.000 and the towns are 12.503; the computational time could be rather long. Selecting only the *focus towns* and only the tourist information we obtained specific results in a reasonable time. We have found 233 *focus towns* with from one to seven tourist attractions that are merged to the data extracted previously.

It could be strange that there is no field to tell us to which town it belongs, but we have found it stimulating. Actually we had to use the Oracle Spatial tools to discover the point of interest affiliations. With a simple *SQL* query, given the two geometries (area for town and point or area for points of interest), we can extract if the area of the town contains the point of interest, touches the point of interest or if they are disjoint. For example we have a point of interest which identify a central station and we want to know in which town. The result of the spatial query will attach a *GKZ* denoting the town membership to the point of interest.

2.6.2 Data extraction from varying geographic units

Some further information about towns was available in a old database of 2002. In this the information of a town is described in seven variables:

- HAUSER(quantitative): number of houses;
- BETRIEBE(quantitative): number of companies;
- ZENTRALITT(qualitative): qualitative value of centrality;
- BODENPREIS_STADT(qualitative): ground price of down-town;
- BODENPREIS_LAND(qualitative): ground price of country-town;
- MIETPREIS(qualitative): rent price;

Such database did not fit the new one because after 2002 there were 2 town divisions and 338 town unions. In figure 2.6.2, considering *case 1*, we can observe an example of a territorial division. In fact on the left a territorial area is covered by two towns labeled by *GKZ 1*, *GKZ 2* and on the right the same territorial area is covered by three towns labeled by *GKZ 1*, *GKZ 2* and *GKZ 3*. In figure 2.6.2, considering *case 2* instead, we can observe the opposite situation: a town union. A territorial area is covered by three towns labeled by *GKZ 1*, *GKZ 2*, *GKZ 3* and on the left the same territorial area is covered by two towns labeled by *GKZ 1* and *GKZ 2*.

In the following we will explain how we handled both town divisions and unions.

Town divisions

Now assuming to be as in figure 2.6.2 *case 1*, the variables of territorial entitity labeled as *GKZ 1* on the left have to be transformed into valid variables for *GKZ 1* and *GKZ 3* on the right. The qualitative variables remain coherent, for instance if $MIETPREIS = 1$ in *GKZ 1* left, the variables respectively of *GKZ 1* and *GKZ 3*

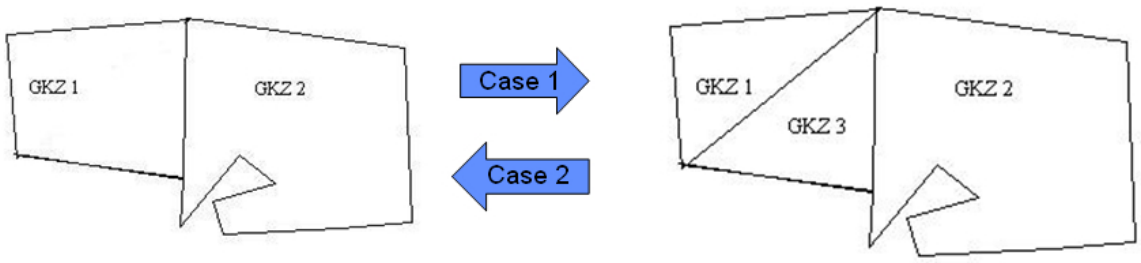


Figure 2.14: Town division and union example

right will be $MIETPREIS = 1$ and $MIETPREIS = 1$. So for the four qualitative variables we just copied the corresponding values.

The variable *HAUSER* is a quantitative variable and the values assumed with a division are wrong. In fact, if we copy $HAUSER = 10.000$ of *GKZ 1* left into *GKZ 1* and *GKZ 3* right, it is absurd that the same territorial area has double number of houses $10.000 + 10.000$. Our approach to tackle this problem has been to compute a weighted average on the number of inhabitants, the resulting output is not completely taccurate, but preserves the consistency. Assuming the value of variable $INABITANTS = 30.000$ for *GKZ 1* left and $INABITANTS = 20.000$, $INABITANTS = 10.000$ for *GKZ 1*, *GKZ 3* right, the two new values of $HAUSER$ will be $10.000 * \frac{20.000}{30.000} = 6.666,66$ and $10.000 * \frac{10.000}{30.000} = 3.333,33$. The same weighted average on the number of inhabitants is computed for the other quantitative variable $BETRIEBE$.

Town unions

For town unions as shown in figure 2.6.2 *case 2*, the variables of territorial entitities labeled as *GKZ 1* and *GKZ 2* on the right have to be transformed into valid variables for *GKZ 1* on the left. To compute the quantitative variables an addition is needed,

for instance if $HAUSER = 100$ and $HAUSER = 150$ in *GKZ 1* and *GKZ 2* right, the variable of *GKZ 1* left will be $HAUSER = 100 + 150 = 300$. Then this step is repeated for the other quantitative variable $BETRIEBE$

Therefore the problem to compute consistent value in this case is actually for qualitative variables. There are situations in which two towns that should be merged into one have different values for a qualitative value, the question is: Which value should I take? We have chosen to take the highest value for *BODENPREIS_STADT* because it is relative to the price of the main center in the area. thus if two towns are unified we will need only the value of the main center and it is more probable the highest one. For the remaining qualitative variables *ZENTRALITÄT*, *BODENPREIS_LAND* and *MIETPREIS* a weighted average is computed on the number of houses, then the resulting value is approximated. Assuming the value of variables $HAUSER = 10.000$, $HAUSER = 15.000$ and $MIETPREIS = 2$, $MIETPREIS = 3$ for *GKZ 1*, *GKZ 3* right, the new value of $MIETPREIS$ will be $2 * \frac{10.000}{25.000} = 0,8 + 3 * \frac{15.000}{25.000} = 1,8$, giving an approximate result of $MIETPREIS = 3$.

Chapter 3

CLUSTERING MODEL

The previous chapter deals with data transformations, abstracting more general information for spatial data and adapting data set structures for non spatial data. At this point we obtained more useful data for our purposes, but before starting to build the clustering model we need to select the variables which explain better the domain and normalize them by the medium of statistical tools. We obtained 2482 observations with following variables:

- *SERVICE_LEVEL*(nominal): output of section 2.3 . It assumes values from 1 to 7.
- *PATH* (nominal): output of section 2.4. It assumes 1 if the town lies in a path between two bigger towns, 0 otherwise.
- *STREET_NET*(nominal): output of section 2.5. It assumes values from 1 to 312.
- *TOURIST*(numeric): output of section 2.6. Number of tourist attractions.
- *EW*(numeric): number of tourist attractions.
- *QKM*(numeric): town area.

- *HAEUSER_05*(numeric): number of houses.
- *BETRIEBE_05*(numeric): number of companies.
- *ZENTRALITAET_05*(nominal): it denotes the centrality of the town. It assumes from 1 (minimum) to 4 (maximum).
- *BODENPREIS_S_05*(nominal): foundation price for houses in the town. From 1 (cheapest price) to 7 (most expensive price).
- *BODENPREIS_L_05*(nominal): foundation price for houses in the town periphery. From 1 (cheapest price) to 7 (most expensive price).
- *MIETPREIS_05*(nominal): rent price for houses. From 1 (cheapest price) to 7 (most expensive price).

In figure 3 for each variable we have the correlation with all the others.

	path	servic~l	street~t	tourist	ew	qkm	kk	hauser	betriebe	zental~t	boden_~s	boden_~l
path	1.0000											
service_le~l	0.0864	1.0000										
street_net	-0.0027	0.0021	1.0000									
tourist	0.0549	0.0770	-0.0072	1.0000								
ew	0.2216	0.1598	-0.0248	-0.0801	1.0000							
qkm	0.0920	0.0286	-0.0264	-0.0591	0.4027	1.0000						
kk	0.1019	0.0302	-0.0089	0.0627	0.1126	-0.2492	1.0000					
hauser	0.2444	0.1313	-0.0289	-0.0663	0.9584	0.4710	0.1081	1.0000				
betriebe	0.1960	0.1462	-0.0246	-0.0801	0.9570	0.3471	0.2204	0.9250	1.0000			
zentaliraet	0.1293	0.1188	-0.0192	-0.0896	0.7182	0.4842	-0.0546	0.7073	0.7058	1.0000		
boden_prei~s	0.1798	0.1381	0.0088	0.0274	0.5388	0.1570	0.3514	0.5346	0.5780	0.6455	1.0000	
boden_prei~l	-0.1231	-0.0839	0.0127	0.0335	-0.4799	-0.3927	0.0715	-0.4977	-0.4681	-0.7978	-0.7581	1.0000
mietepreis	0.0114	-0.0019	0.0135	0.0078	0.1435	-0.1042	0.4141	0.0944	0.1902	0.0012	0.2161	0.0878

	mietep~s
mietepreis	1.0000

Figure 3.1: Correlation between variables

The curse of dimensionality is the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space. It is a problem which could obstacle the success of a good model. For this reason we decided to reduce dimension of our mathematical space dropping the variables *BETRIEBE* and *HAUSER* which have a correlation equal to circa 0.95 with the variable *EW*.

The numeric variables have different unit of measurement, therefore a normalization to make them homogenous is needed. The remaining variables are 11, of which 3 numeric normalized and 8 nominal.

In the following we give the results of the two clustering algorithms, *EM* and *K-MEAN*.

The Data Mining tool *Weka* offers an easy way to compute clustering algorithms for text files containing the data set. For the *EM* algorithm is sufficient to specify the variables to be computed. At this point the results of the *EM* clustering groups the towns in 13 clusters a thematic layer identifying the cluster memberships is shown in figure 3

The results of *K-means* algorithm is shown in figure 3, for the number k of clusters is chosen the best number of clusters (13) found in previous section with *EM* algorithm.

3.1 Evaluation

For all clustering algorithms in this thesis we have done a deep analysis and evaluation of results. The evaluation of the final clustering is not part of our competences. In fact at the moment we are not able to evaluate such models because we have not the statistical tools for frequency computations to do it. We showed the best models according variance infra and inter clusters, but it could be not be best one for computing of street segment frequencies. For example the measured frequencies available could not cover all clusters. We remember that we infer a frequency of a street segment by computing comparable street segments in comparable towns. In this case the inferred frequencies could not be cover all towns because we have not at least a measured frequency for each cluster. So a solution could be to change the input number of clusters to the clustering algorithm.

A complete evaluation could be possible only at the end of *frequency map 10-50* project.

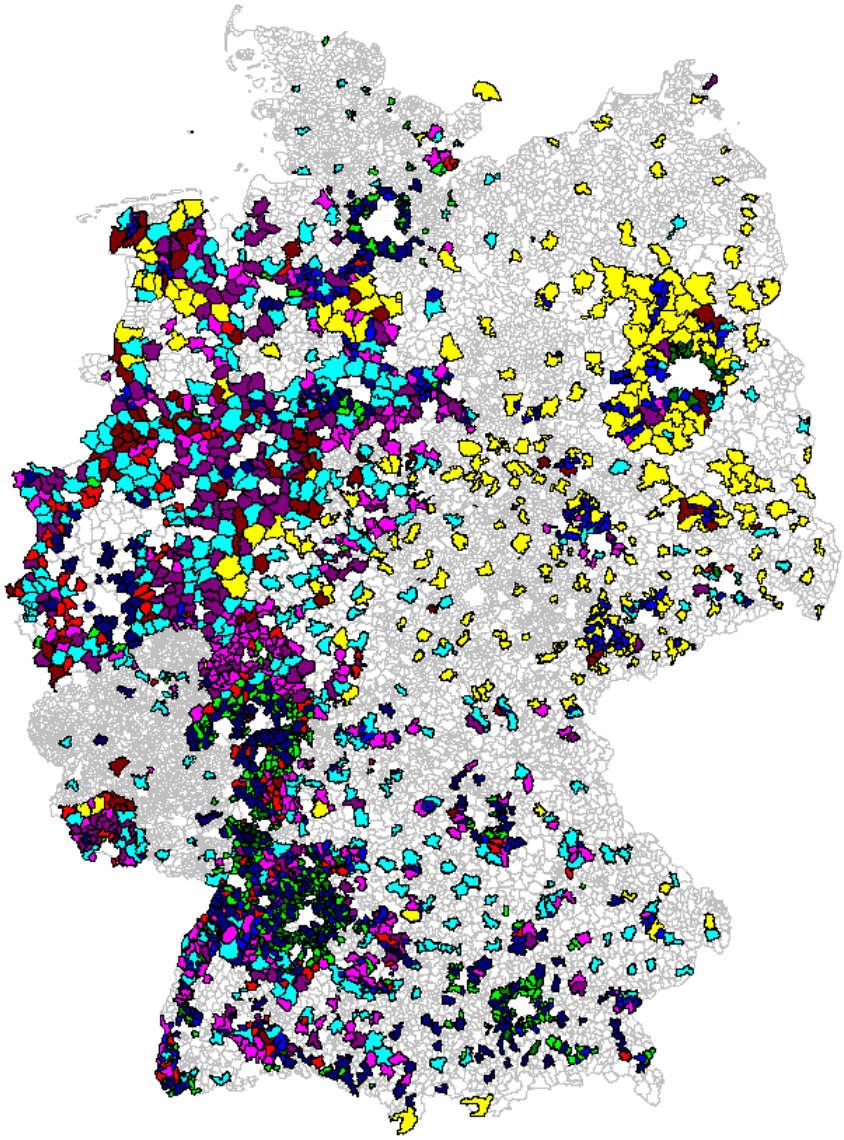


Figure 3.2: Results of the *EM* algorithm

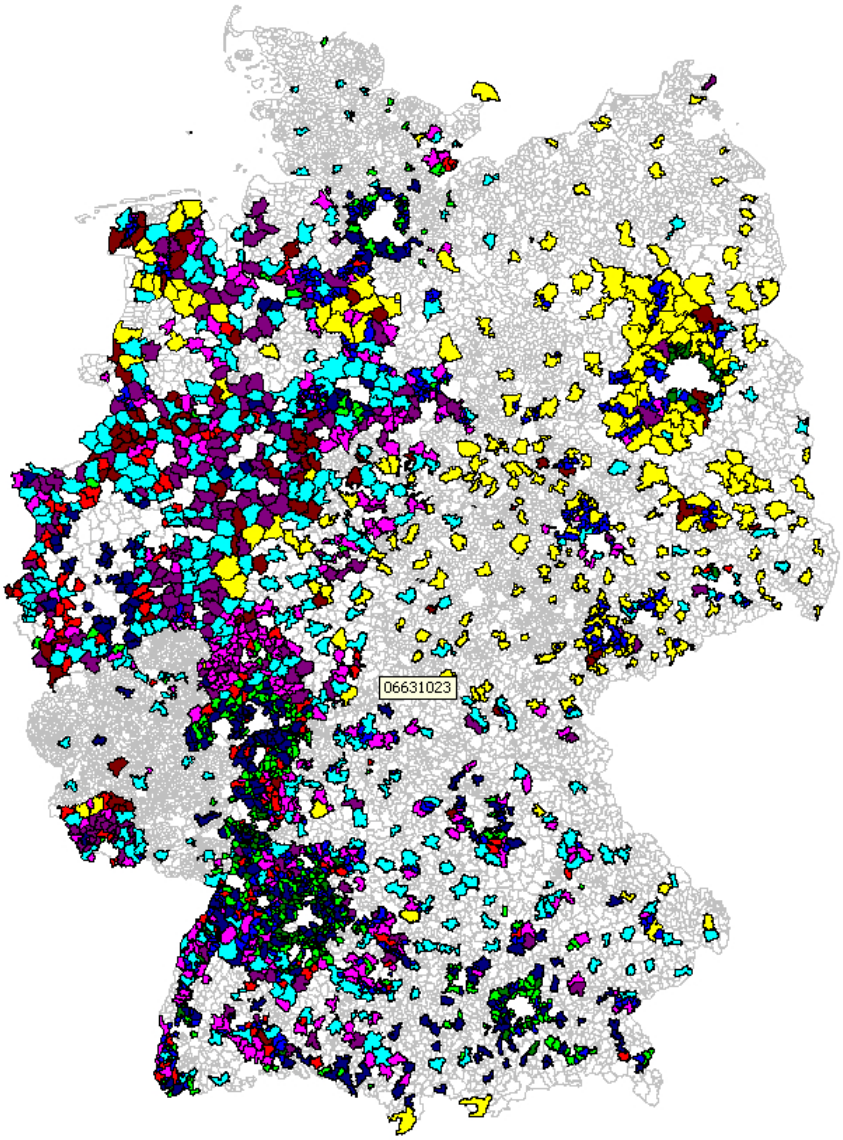


Figure 3.3: Results of the *K-means* algorithm

Chapter 4

CONCLUSIONS

In all the thesis we described each task in a sequential order, here we invert this order: we start from the obtained clustering model and we go backwards.

Each of the obtained 13 groups of towns has at least a town with measured frequencies of its street segments, we call it for simplicity *measured town*. The *measured towns* are the basis of the statistical model which infers the frequencies of comparable street segments in *comparable* towns (in the same cluster). Imagine to focus our attention to a single cluster. Taking a random street segment x with frequency c belonging to a *measured town*, equal frequency c to all comparable street segments in the cluster are assigned. We think it is a good way to proceed. Anyway we ask why should we have good results? Or better, why the towns in a certain cluster should have similar frequencies? The principle base of a clustering algorithm is that the similarity within a cluster is maximized. The similarity is in term of the values of the variables forming the data set. It means that the number of tourist attractions and the value of other non-spatial information should have not a big variance. This support our hypothesis, for example if the towns have the same number of tourist attractions, they could attract the same number of tourists and create a certain traffic on their streets. It is valid for the other variables concerning the spatial information too. The *service level* value should be more or less the same, therefore it could mean for example that the tows in such cluster could be most "isolated towns", theoreti-

cally with low frequencies. The street network structures should match each other, they should have similar street pattern, radial for example. They should have most the same *path value*, for example not being in between two *big towns*, so having lower frequency. We think that this kind of reasoning support our purposes, in fact the results has been successfully applied in practice.

The SCOT's core is the spatial data preparation, how we make explicit the spatial information. There are ready and successful algorithms to threat a spatial clustering in which the spatial contiguity is a constraint. But they did not fit our problem, we wanted that two towns in two opposite places in Germany could be grouped in the same cluster. SCOT in substance deals with the special case of German towns, but it could be extended to general objects, exploiting the structure of the object and its relationships with other objects where the contiguity is not fundamental. For example we want to cluster buildings. After making explicit their structures, shape and plant, we will discover their relationship with other objects: the building y is surrounded to all sides by a park. SCOT could be applied to all the cases in which we need to cluster spatial objects and their structure and their relationships are more important than the localized place in which they lie.

List of Figures

List of Tables

Bibliography

[1] Arantza Casillas, Mayte Teresa González de Lena, and Raquel Martínez. Document clustering into an unknown number of clusters using a genetic algorithm. In *TSD*, pages 43–49, 2003.

[2] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[3] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[4] A. T. Estivill-Castro, V. Murray. Mining spatial data via clustering. Technical Report FIT-TR-1997-05, 11, 1997.

[5] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. In *Knowledge Discovery in Databases*, pages 1–30. AAAI/MIT Press, 1991.

[6] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of data mining*. MIT Press, Cambridge, MA, USA, 2001.

[7] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.

[8] B. Hofmeister. The study of urban form in germany. *Urban Morphology*, 8:3–12, 2004.

[9] J. Han M. Kamber and A. K. H. Tung. *Spatial clustering methods in data mining: A survey in Geographic Data Mining and Knowledge Discovery*. New York, 2001.

[10] J. Lafrenz. Bewertungszyklen vorindustriller stadtgestalt im industriezeitalter. *Die alte Stadt*, 16:39–57, 1987.

[11] Donato Malerba, Annalisa Appice, Antonio Varlaro, and Antonietta Lanza. Spatial clustering of structured objects. In *ILP*, pages 227–245, 2005.

[12] Harvey J. Miller and Jiawei Han. *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, Inc., Bristol, PA, USA, 2001.

[13] M. Nanni and S. Rinzivillo. State-of-art of spatial and spatio-temporal data mining. 2005.

[14] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[15] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[16] S. Openshaw. Two exploratory space-time-attribute pattern analysers relevant to gis. In *Spatial Analysis and GIS*, pages 83–104, 1994.

[17] Friedrich Ratzel. *Die geographische Lage der grossen staete*. 1903.

[18] A. Rosenfeld. Image analysis and computer vision: 1997. *Computer Vision and Image Understanding*, 70(2):239–284, May 1998.

[19] Wikipedia. Central place theory — wikipedia, the free encyclopedia, 2007. [Online; accessed 6-luglio-2007].

[20] Wikipedia. Computer vision — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-luglio-2007].

[21] Wikipedia. Depht-first search — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-luglio-2007].

[22] Wikipedia. Graph labeling— wikipedia, the free encyclopedia, 2007. [Online; accessed 10-luglio-2007].

[23] Wikipedia. Graph matching — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-luglio-2007].

[24] Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. ACM Press, New York, NY, USA, 2002.