Algorithms for flow maps

van de Ven, B.M.F.J.

Award date:
2008

technische universiteit eindhoven
Department of Mathematics and Computer Science

Master's Thesis

Algorithms for flow maps

by
B.M.F.J. van de Ven

Supervisor
Dr. B. Speckmann

Eindhoven, December 17, 2007

Abstract

Flow maps are a specific type of map used to depict interaction or movement between regions (e.g. migration of people, goods, etc.). Each flow representing such a movement is visualized by an edge with thickness corresponding to the value of the flow connecting the associated pair of interacting regions. In this thesis we explore algorithms to draw flow maps by automated means depending on aesthetic criteria such as vertex-edge distance and angular resolution. We show that optimizing these maps only for vertex-edge distance is *NP*-hard already, and therefore explore heuristics which can generate good flow maps in general.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to acknowledge and thank my graduation supervisor Bettina Speckmann for all the comments, ideas, discussions and critical reviews that helped to improve this thesis. Furthermore I wish to express my gratitude to Jack van Wijk and Alexander Wolff for being in my examination committee. I would also like to thank my officemates for their additional help Joost, Chantal, Dennis, Remko, Ludo and Arjan. Lastly, I'd also like to thank my friends and my family.

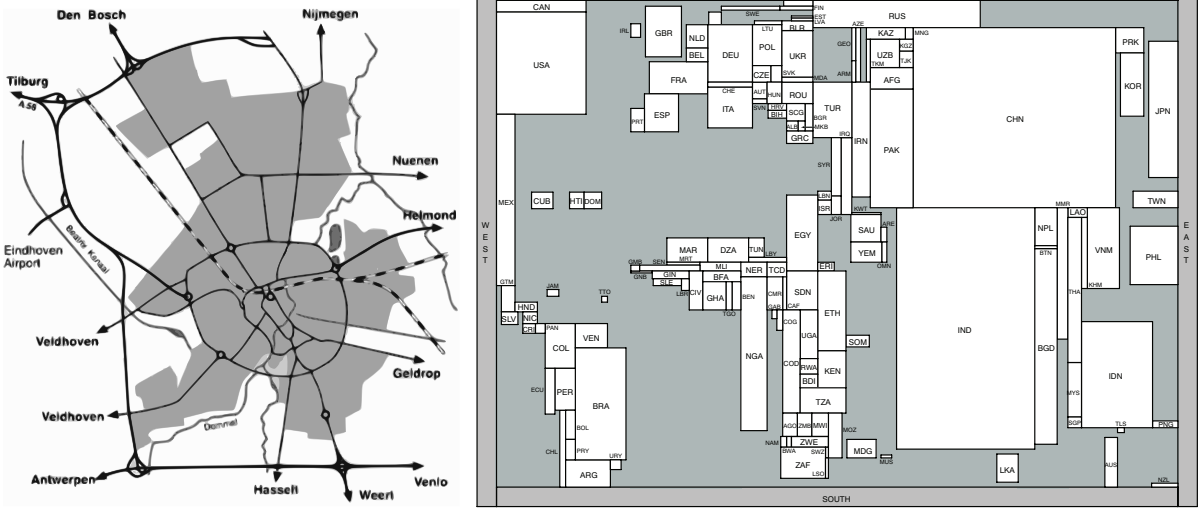
Bas van de Ven

Eindhoven
December 17, 2007

Chapter 1

Introduction

In the field of cartography maps are generated to depict specific characteristics of spatial or geographical data about certain regions visually. A clear distinction can be made between two types of maps, namely: *geographic maps* and *thematic maps*. Geographic maps are the most well known type of maps as they can be found almost anywhere, from atlases to informative signs next to the road at a town's entrance. They display the geographical features of the region (e.g. cities, country borders, streets, rivers, etc.) accurately and often with much detail. An example of a simple geographic map of the city of Eindhoven in the Netherlands can be seen in Figure 1.1(a).



(a) Geographic map of Eindhoven [1].

(b) Thematic map of the world scaled by actual population per country shown as rectangles generated by a computer [34].

Figure 1.1: Geographic and Thematic maps.

Thematic maps on the other hand visualize data of certain characteristics of the region in a geographical context. One can think of many different data sets including, but not limited to: population, occurrences of tidal waves, inflation, life expectance rate or migration of people. Several types of thematic maps exist including some mixed variations. Each type has its own properties and therefore certain data sets can be shown more succinctly using a specific type as shown in the following sections. An example of a thematic map is shown in Figure 1.1(b).

1.1 Thematic maps types

All thematic maps have in common that they use a geographical *base map* showing the regions by their boundaries. Essentially these are the contours of the landmasses and the internal borders separating the individual regions on them. In some cases the information of the data set is used to modify this map accordingly.

The most commonly known type is the *choropleth*[1] map. This kind of map fills each of the regions on the map using one out of several shades of the same color to denote the value for the corresponding data value. As expected intuitively a light color means a low value and a darker shade corresponds to a higher value. Data sets which are often used for this map type are of a single variable such as data sets for population. Such a specific one-variable thematic map is also called a *cartogram*. See Figure 1.2 for an example of such a choropleth cartogram. It can also be done by applying different visualization methods. One of them is changing the size of the regions corresponding to the data values. If this mapping is used the cartographer does need to take care that the generated map is still recognizable. The size modification may cause some areas to completely vanish from the map, or conversely they could take up so much room that the rest of the map becomes invisible. The map in Figure 1.1(b) is an example of such a cartogram generated using a computer by Speckmann and van Kreveld [34].

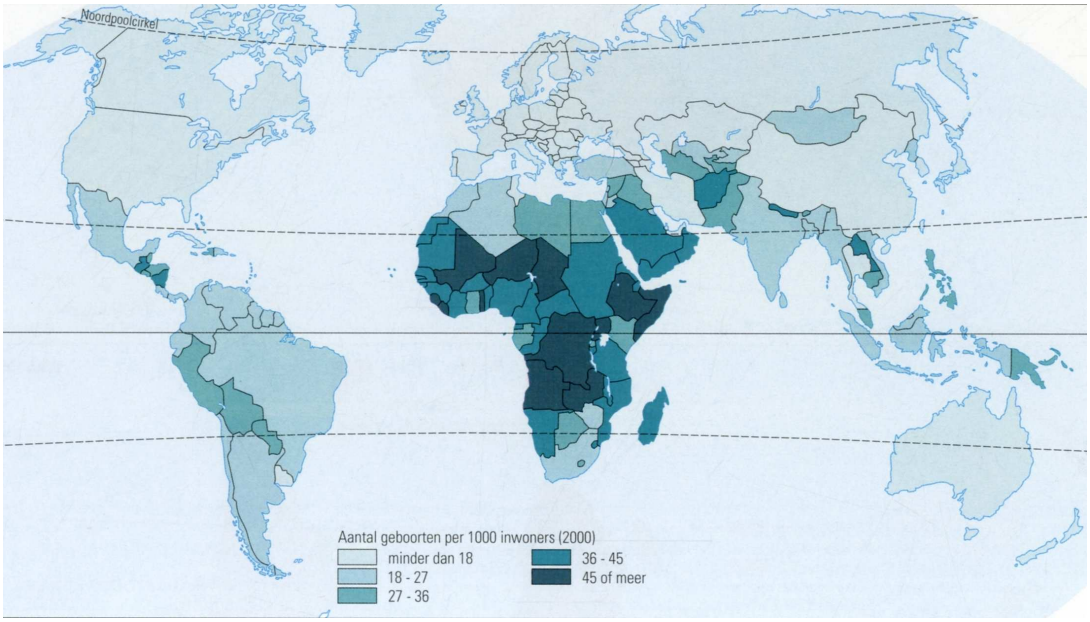


Figure 1.2: A choropleth map depicting birthrate per country in the world in the year 2000 [36].

1.2 Overlay and variations

Some data sets cannot be drawn nicely on the base map itself or if modification of the base map is unwanted in the first place an alternative solution must be found. In these cases an overlay can be used to draw the required information on top of the base map. Such a layer is called a *thematic overlay*. Several types of these maps can be seen in Figures 1.3 and 1.4.

For each of these map types there are two main quality criteria to which they must comply in order to be aesthetically good map. Firstly, the base map should be clear and concise as it would otherwise only distract the reader from the important information of the map. And

[1] in the literature this is sometimes mistakenly spelled as chloropleth map as people falsely assume that it is derived from the Greek word for color.

secondly, the overlay has to avoid overlapping either too much or specific *critical features* of the underlying base map. With critical features one can think of distinct parts of the region and its boundary which are its trademarks that make them easy recognizable for humans. These include three country points, bodies of water, and the borders themselves. In all three cases one wishes to minimize the coverage of these parts of the maps as much as possible.

Labeled map. If one has a data set on the locations of, say active volcanoes, in the vicinity it would be fitting to show them on the base map as little red triangles. Hence creating an overlay with these icons depending on their location generates a *labeled map*. Labeled maps occur frequently in combination with other map types as adding additional (textual) labels can often clarify the orientation and position for the viewer.

Proportional symbol map. Data sets consisting of quantities of more than one parameter like the production of several types of ore per country can be depicted by labels too. But since these values represent individual quantities it would be better to generate symbols which are proportional to their value. By doing so it allows the viewer to compare the differences per type in a single view. This type of visualization gives rise to the so-called *proportional symbol maps*. One of the familiar examples is the pie chart which shows distribution of the parameters for each individual region within the pie. The size of the pie itself shows the relative values between the regions themselves. Figure 1.3(c) displays a map about the types of employment in the Netherlands as a proportional symbol map.

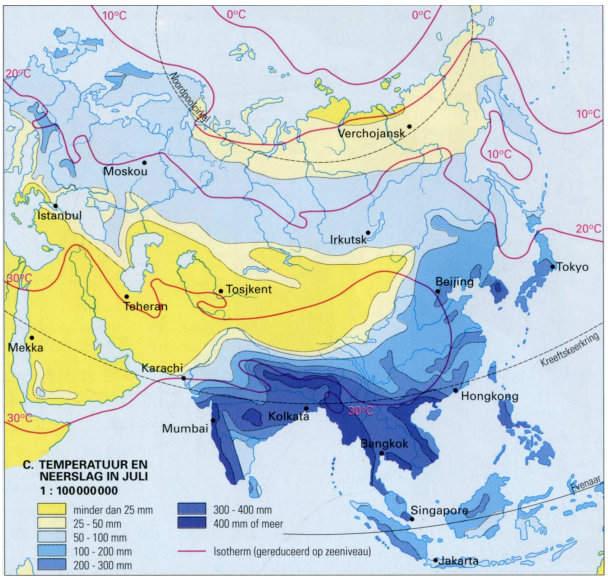
Dot map. Other possibilities include *dot maps* where each dot represents a certain value for that specific area in the region, thus the intensity of dots depicts the proportional value for that region. A setting which fits the use of this kind of map are the ones showing exact locations for specific types of agricultural use of the land. These are often very local of nature, because of the way the environment is built up. This often results in clusters at specific areas of the region, which fits this method of visualization properly. An example of this map type can be seen in Figure 1.3(b).

Isoline map. Isolines can be used for example to depict temperature or air pressure. These values are not bound to the borders of the countries and therefore the use of an additional layer shows these lines more adequately. This leads to the creation of an *isoline map* as shown in 1.3(a). This particular map is of the mixed kind as it shows the temperature in isolines, some major cities as dots with textual labels and the base map is a choropleth map depicting the quantity of rainfall.

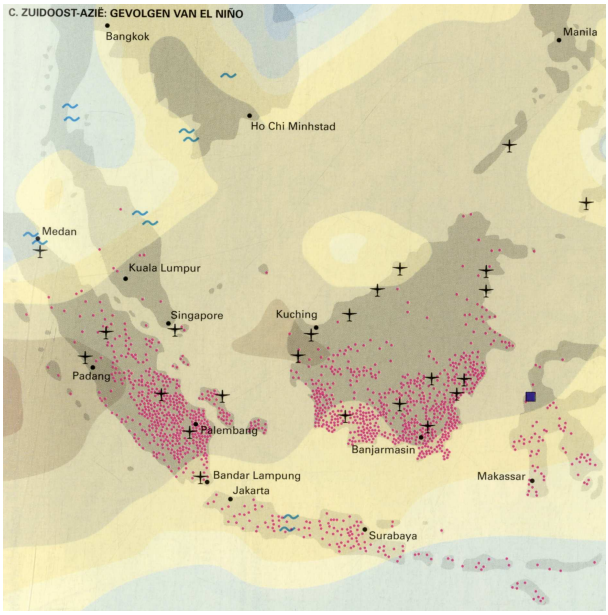
Flow map. Maps which display movement or interaction between places are hard to visualize with any of the aforementioned techniques as those methods only represent the data locally (e.g. a symbol, color, etc.). In order to overcome this problem these data sets can be visualized with the aid of a so-called *flow map*[2]. Examples of typical data sets for these are migration of people between certain places, the exportation of red wine from one country to others and the movement of companies within the Netherlands. The last two examples are shown in figures 1.4 and 1.3(d).

Depiction of the flows in a flow map is done by drawing lines on top of the base map between the corresponding locations which represent the flow from one region to another. The color intensity and/or thickness of the flow depict the volume which is proportional to its corresponding

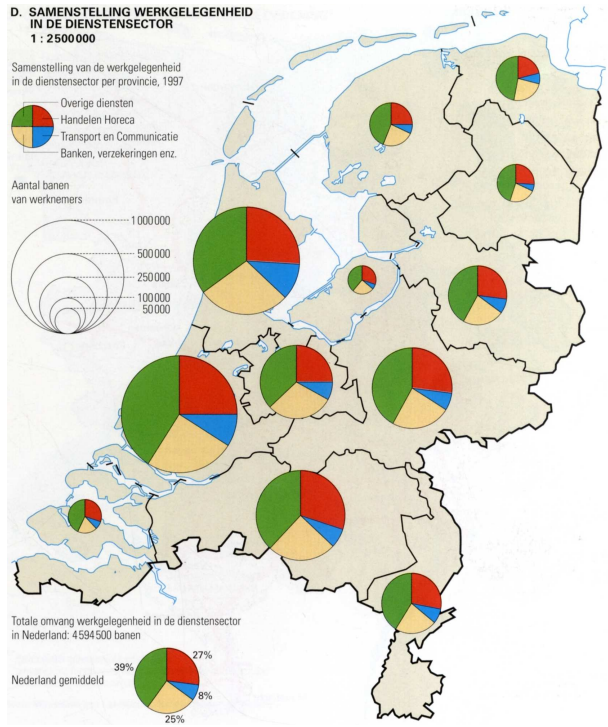
[2]also referred to as *dynamic map* in some cases



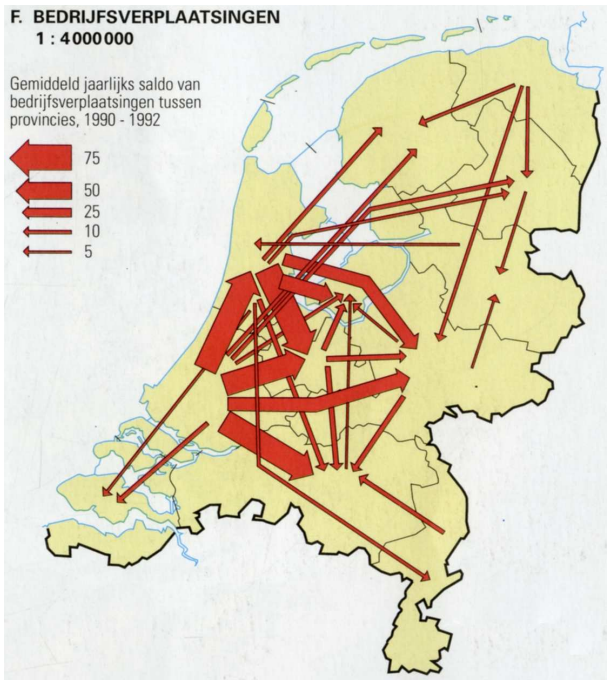
(a) A mixed isoline map showing average temperatures in July, the quantities of rainfall as a choropleth map of the same month, and some of the important cities with textual labels in most of Asia [36].



(b) A dot map showing the showing the locations of great fires after El Niño hit parts of Indonesia in September 1997 [36].



(c) A proportional symbol map showing employment of people in pie charts [36].



(d) A flow map on the movement of companies in the Netherlands [35].

Figure 1.3: Four distinct types of thematic maps with overlay taken from the Bos Atlas [35, 36].

value. This can be done by translating the value using a linear[3] function to line thickness, and also by classifying the values to between certain ranges (5 to 7 ranges is the most common). In most cases these lines are strait, but there also exist maps which use arcs or polylines to draw them. Arrowheads may be used to show the actual direction of the flow's movement if required.

1.3 History of flow mapping

Flow maps originate from back in the late 18^{th} century. At that time Henry Drury Harness and Charles Joseph Minard were among the first to make these type of maps [11]. Their motivation for creating maps with flows was that they wanted to show large amounts of numerical data about migration in an easy to read format. Instead of using large tables with many numbers a visual depiction on the map could easily explain the special characteristics of the table in an instant. They drastically simplified the underlying map to attract the reader's attention to the most important parts (namely the flows themselves). Figure 1.4 shows a flow map drawn by Minard. It doesn't require much insight to see that the Straits of Dover and countries such as

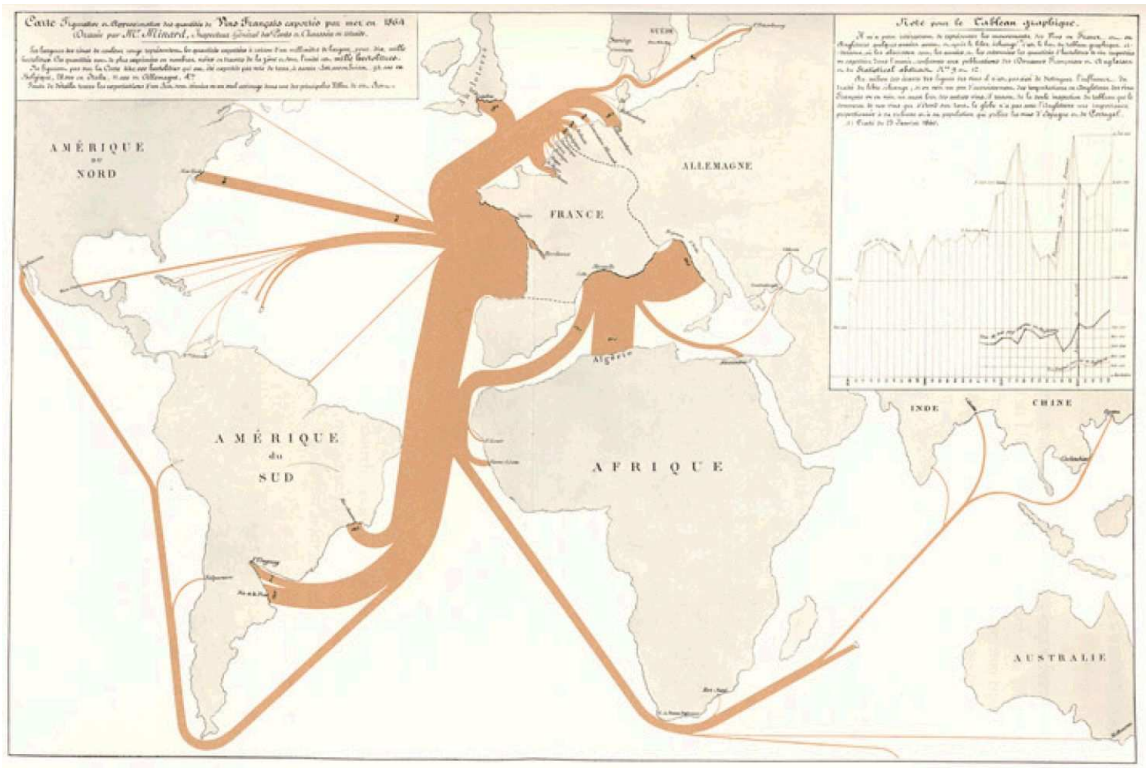


Figure 1.4: A flow map drawn by Minard on the exportation of red wine from France to other countries in the world in the year 1864 as seen in the book by Tufte [33].

Spain are deformed vastly in order to make the flows fit and preventing overlap with the land. In this case however such deformations are acceptable as it improves the map aesthetically and the result is recognizable without difficulty.

1.4 Flow mapping with computer assistance

The problem of drawing flow maps with the computer relates to *graph drawing* in the area of *geometric algorithms* within the field of computer science. One of the main topics which is

[3]Tobler [31] also reasoned about using exponential or logarithmic functions.

researched in this area is about drawing a graph visually optimized for one or more *aesthetic criteria*. One can think of several criteria such as vertex-edge distance, angular resolution, number of edge crossings and uniform edge length to name a few. One specific approach in this research area is *force-directed* graph drawing. The name force-directed comes from the analogie in physics, because one assigns repulsive forces to the edges and vertices which build up the graph. The edges can be treated as springs which exert force $\mathbf{F}$ conform *Hooke's law*:

$$\mathbf{F} = -k\mathbf{x}, \tag{1.1}$$

where $\mathbf{x}$ is the distance the spring has been stretched or compressed from its equilibrium position in meters, $\mathbf{F}$ is the restoring force exerted by the material in newtons, and k is the *spring constant* or force constant which is defined for a spring in N/m.

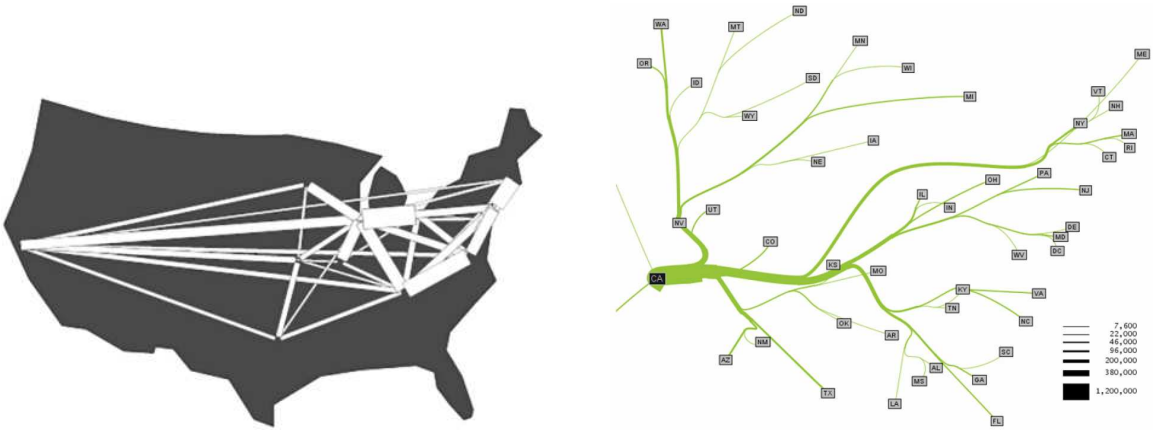
The vertices move similarly to charged particles using the law of *Coulomb* which can be stated for two particles with charge q_1 and q_2 separated at distance r. The repulsive force $\mathbf{F}$ in vector form becomes:

$$\mathbf{F} = \frac{1}{4\pi\varepsilon_0}\frac{q_1 q_2}{r^2}\widehat{\mathbf{r}}_{21}, \tag{1.2}$$

where ε_0 is the constant for vacuum permittivity and $\widehat{\mathbf{r}}_{21}$ is defined as the unit vector parallel with the line from charge q_2 to charge q_1. When all forces are defined the graph is simulated as if it is an actual physical system, until an *equilibrium state* has been reached. In this state there is no movement because all of the forces cancel each other out leading to the equilibrium which is the final drawing. Several papers each using their own distinct methods have be written on this topic. The fundamentals of this graph drawing method and several references to related work in this area are treated in the book by Battista et al. [3]. This approach is used for the implementation in this thesis as to be defined in more detail in the following chapters.

1.5 Related studies

The topic of computer-assisted flow mapping is far from extensively researched. The best-known flow mapper to date is the one by Tobler [29]. This mapper produces generic maps without any optimization with respect to the visualization of the data. A picture from the mapper's output can be seen in 1.5(a). There are two main problems with the mapper which could be improved.



(a) Data set from Table C.1 mapped by Tobler's flow mapper [29]

(b) Output map from the mapper by Phan et al. [21] It shows the migration from California between 1995-2000.

Figure 1.5: Different types of flow map generation by computer.

The first problem is that the algorithm which generates the thematic layer doesn't take any

features of the base map into account. As explained earlier, the obfuscation of certain parts of the areas can make it harder for the viewer to understand the map. In data sets with huge deviations it can cause poorly readable maps, because the arrows may become unforbiddingly wide covering complete areas or other flows. This, however, is a data scaling problem and falls outside the scope of this thesis.

Another flow mapper exists created by Phan et al. [21]. This mapper generates flow maps that are somewhat different from the conventional type as the mapping algorithm distorts vertex positions in order to improve the readability of the entire picture. For some applications this is no problem, but for our setting it leads to undesirable results. In general it is preferred to leave the base map as it is, because the map is better understandable to the viewer if the relative positions of the regions are maintained. An example of their mapper's output is depicted in 1.5(b).

1.6 Thesis overview

In this thesis new methods for the automated generation of flow maps are treated. First of all, the cartographic issues and the aesthetic criteria for flow maps are considered in Chapter 2. This is required as the problem needs to be defined properly before attempting to create an algorithm for the mapper. The complexity analysis of vertex-edge distance problem is covered in Chapter 3 which also shows *NP*-hardness for the two-dimensional version. Chapter 4 describes how the criteria can be transformed to an algorithm which optimizes the flows on the map. After that the implementation and the test results of various settings are shown and compared in Chapter 5. These tests are carried out to see which configuration gives the most aesthetically pleasing maps. Afterwards the conclusions, enhancements and ideas for future improvements are described.

Chapter 2

Quality criteria for flow maps

In this chapter the in- and output for our mapper are defined along with the criteria our output has to comply with. We are given a base map consisting of n regions and an input table of size $n \times n$ containing all the input values in a from-to format corresponding to each of the regions on the base map. The required output is a flow map which shows the data from the table on top of the base map in an aesthetically pleasing way. By the definition of a flow map each edge drawn on the map will be associated with the two regions which are its begin- and endpoints. Therefore these points must be inside the regions they correspond to. One can allow these locations to be anywhere inside the region, but for this thesis we constrain ourselves to associating all edges connected to a region with a single vertex. This vertex acts as the begin- or endpoint for all the edges connected to that region. Alternatively, one could choose to use distinct endpoints within the region for each edge. Furthermore we restrict ourselves to edges that are visualized as straight line segments with arrowheads to denote direction in stead of using polylines or curved lines. The critical features of the map are defined as the three country points, and some additional specified regions (internal bodies of water) on the base map which should not be occluded by the flow lines. This also includes that the flow lines should try to minimize the amount of overlap with the region boundaries. Next to that the output must comply to each of the quality criteria given below in order to be readable and aesthetically pleasing.

Several references in the literature are used to define the quality criteria for flow maps in general. The best resources can be found in chapter 12 from Dent's book *Cartography: Thematic Map Design* [11], the research of Ravenstein [24, 25], the work of Tobler [30, 31, 32], and the Master's thesis by Parks [19]. However, some of these criteria such as the placement of the legend and similar aspects do not apply to this study. A few others such as vertex positioning need to be refined as generating a map with a computer requires well-defined metrics for which we can optimize the map.

2.1 Clear and concise base map

As information of a flow map is in the thematic layer the base map should not show too many details as it distracts the reader from the most important parts of the map which are the flows themselves. Hence the use of simple polygons which follow the bounding contours for each of the n regions is sufficient to display the base map adequately. The polygons can be filled with a single color in order to make them stand out from the background, but at the same time these regions on the base map should not attract too much attention. Therefore it was chosen to use a light to medium tone of grey for filling the regions with a background color.

2.2 Suitable data set to visualize

Given the square input matrix with n^2 values an equal amount of edges have to be displayed as the movement from region A to region B is not the same as its reverse. If one would naively do such a thing the output is hardly readable if the graph has more than a few vertices. The example in Figure 2.1 shows the complete graph in only a single direction for all 48 contiguous

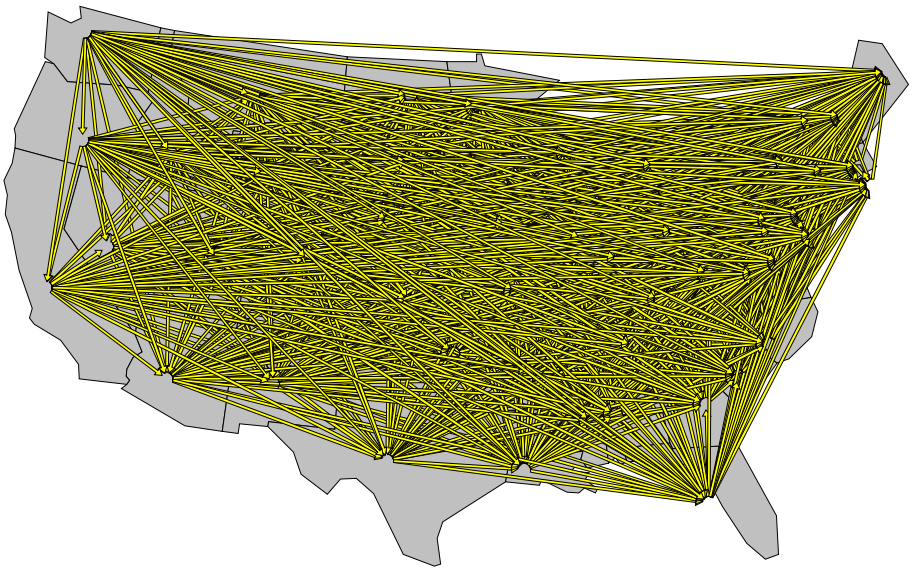


Figure 2.1: Visualizing a dense or complete graph as a flow map always leads to unreadable maps.

states of the USA. It doesn't require any additional argumentation that flow maps can only be of any good if the number of edges is limited. Ideally, a sparse, near-plane graph is what one wants to get as it is clear to draw. To get from an arbitrary data set to a reasonably sparse graph edges need to be removed. Tobler [31] states that the small flows only contribute very little information to the total view. Hence by applying a threshold on the data set all edges below a certain data value will be removed. Experiments by Tobler in the same paper show that choosing a threshold which cuts out approximately 2/3 of all the edges works in general. The resulting graph is less cluttered and it decreases the amount of edge crossings significantly. Cutting away this many of the thinner edges also makes the larger, more important flows stand out better.

2.3 Proper depiction of the data

Each of the edges that needs to be drawn in the graph is associated with a value from the input table. To represent that value the edge thickness is proportional to it. One can apply a function to translate this value to the actual thickness in either a linear, quadratic, or exponential fashion. Linear seems most likely to work out, but when edges get wider the human eye has difficulty to determine which edge is thicker. Hence one might wish to choose for a faster quadratic or even exponential growth function. This however doesn't work out as the large values in the data set become too wide to show on the map even for small numbers. Therefore a linear function is the only practical solution, but with many edges that have a similar value there will be several cases where two distinct edges seem to have an identical width. This makes it hard to notice any difference between them.

Instead of trying to show the difference for each individual edge a classification can help

improving the overall readability of the map. The classification goes at the expense of the exact value the edge represents, but graphs in general are made to show interesting characteristics of data sets they show. A linear classification with 5 to 7 classes is most suitable as adding more classes decreases the benefits of the classification, which is counterproductive. To make the difference stand out even better the edges for each class receive a color. An effective and intuitive way to do so is by using a single color with a variable shade from light to dark depending on its thickness of corresponding class. Picture 2.2 shows the five classes used for our mapper.

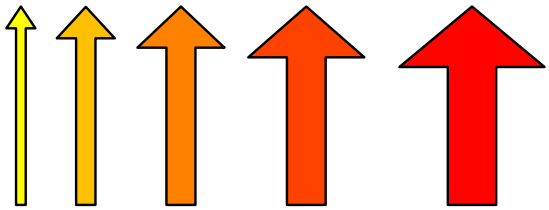
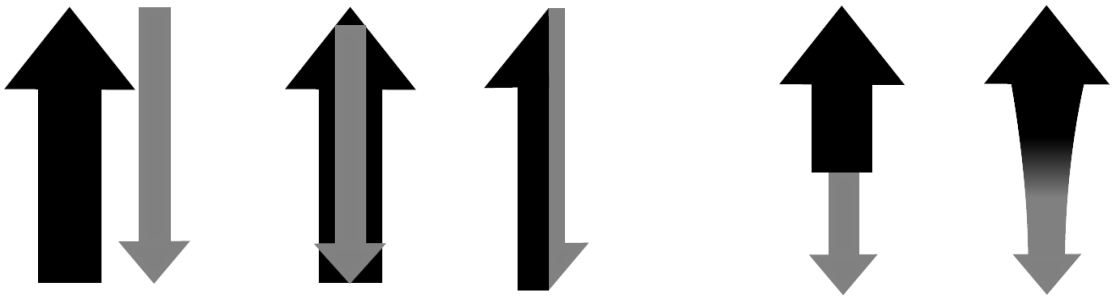


Figure 2.2: The five classes for edge thickness.

The type of arrowheads to use when direction is required has to be considered as well. Tobler did some investigations on which types are aesthetically pleasing in [31]. The possible variations he came up with are shown in Figure 2.3(a). All three have the problem that the view of



(a) Bidirectional edges by Tobler.

(b) Bidirectional edges by us.

Figure 2.3: Several candidate variations for bidirectional edges.

the begin and endpoints must not be obstructed by any other edges or else the direction of the flow becomes ambiguous. Furthermore some additional types are created for this study to contemplate any other possibilities. Two of the most useful ones are depicted in Figure **??**. These two both have the drawback that the width of the arrow might be impossible to determine if critical parts are covered by other edges. In the end the double edged harpoon arrow as shown in the center of Figure 2.3 is chosen as it is the arrow uses its space most efficiently. The only thing one has to bear in mind is that the arrowheads must remain visible when the edges are drawn on the map.

2.4 Edge ordering

Intuitively one would indeed agree that thin lines have to be put over fatter ones as the former ones would hardly be visible on the map otherwise. Tobler [31], however, states that it is better to do it the other way around. As counter intuitive as it may seem experiments lead to the same results as the ones Tobler found. They show that maps become better readable if the most important lines (namely the biggest flows) have the highest priority. An argument in favor for this approach is that thin lines only contribute very little to the overall view, whereas a huge

flow attracts much of the viewer's attention as they appear to be more relevant. Apart from that the map becomes very cluttered if many thin edges are placed over the fatter ones, which is usually the case as the edges with the lowest width class appear the most. See for an example Figure 2.4. In this figure the two images at the top show the same map, but in Figure 2.4(a) the edges are sorted in descending order and in Figure 2.4(b) they are sorted in ascending order of thickness. The image of Figure 2.4(b) seems more clear at the first glance as there is much less cluttering.

Unfortunately, the arrowheads of the thinner edges have gone missing making the map less readable as it becomes harder to see where the smaller edges are going towards. In order to overcome this the edges receive additional shortening depending on their class weight as shown in Figure 2.4(c). This map shows all of the edges again in order of ascending thickness, but now the thickness of an edge also determines how much the edge gets shortened. As visible this technique allows one to see all edge heads and tails without losing the advantages of ascending thickness. This situation is still not optimal as the shortening itself however should be dynamic. There could be large gaps when there are only a few thick edges ending at the same vertex. To take this into account an edge only receives shortening if there is another edge coming into the same point at a similar angle. Hence shortening depends on the size of the smallest angle between the two edges. This type of shortening for the same situation is shown in Figure 2.4(d). By comparing all four images it can be observed that applying some form of shortening in general leads to better pictures as the edges take up less space in the graph.

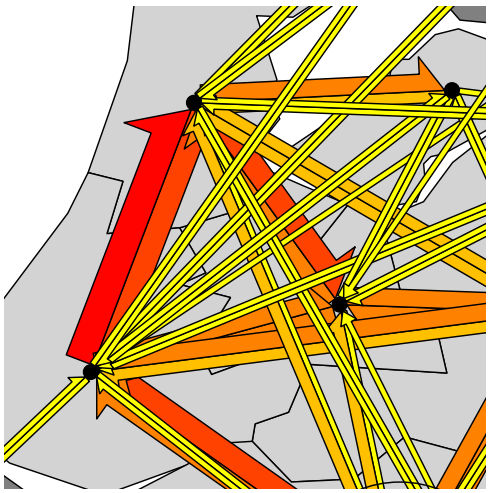
Even though the previous method solves arrow head overlap for the edges coming together at that vertex it does not solve this problem for all cases. Suppose an edge d has its arrowhead obfuscated by any other arbitrary bypassing edge e, then the direction of the arrow cannot be determined on that side of the edge. One could try to raise d to be over e, but then e might cause problems with another edge. This issue is referred to as the *weaving* of edges. Resolving it in this case can either be done by moving the vertex associated with the end of the edge d away from e or by drawing only the obfuscated head of d on top of e and leaving the rest of d as it was. This is also shown in Figure 2.4(e).

Lastly, if we look at the ordering of the incoming edges at a vertex it becomes notable that a proper ordering of edges of the same thickness class (counter)-clockwise around the vertex can improve the visibility of the edges as well. By similar means as for the issue with weaving we can improve this situation by raising only part of the edge head that was obfuscated. Figure 2.4(f) shows a picture of the same area where the edges have been sorted in ascending order received dynamic shortening. Higher priority has been given to edge heads and edges have been ordered around each vertex.

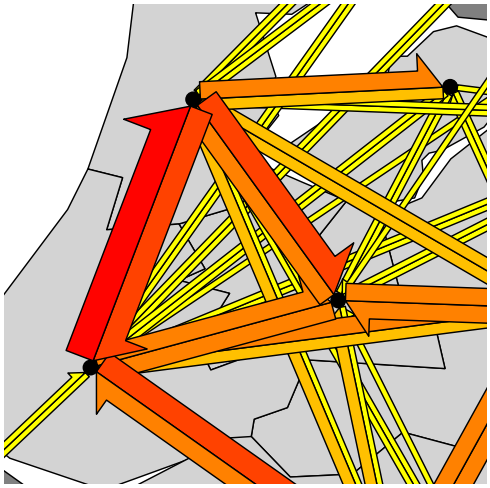
2.5 Minimizing overlap with critical features

Suppose the view of a critical feature cf is blocked by the edge $e = \{v_1, v_2\}$. Then the situation can only be improved by moving edge e away, because cf is part of the base map and thus cannot be moved around. Improving the visibility of cf occurs by moving either one of the vertices in the direction that is perpendicular to the axis of the edge between them. If feature cf is a point then a small movement will suffice as the edges are relatively thin in width. In the case that cf is a shape such as a polygon it is possible that the region boundaries of v_1 and v_2 can not always be positioned in such a way that the feature is completely visible. The only thing which can be done is trying to minimize overlap as much as possible. Next to explicitly defined features the boundaries of the regions themselves are also important to take into consideration, hence they should have minimal overlap as well.

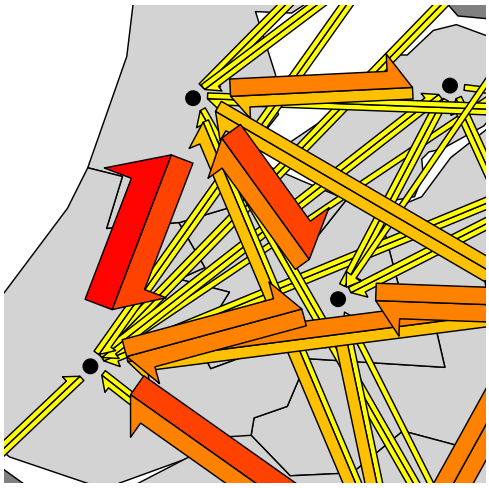
Other techniques to minimize the overlap are using edges with bends in them or curved edges. Applying and implementing such edges are not taken into consideration for this thesis, but it



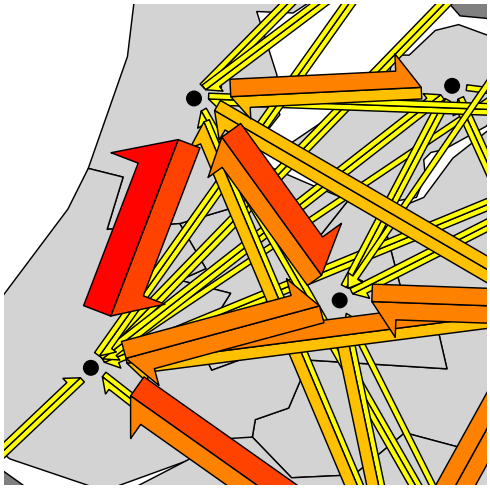
(a) Sorted descending in thickness.



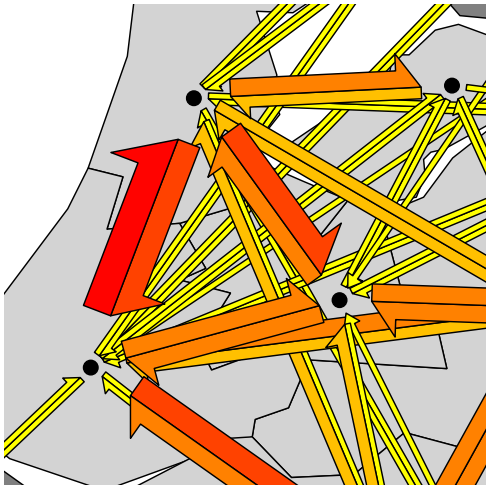
(b) Sorted ascending in thickness.



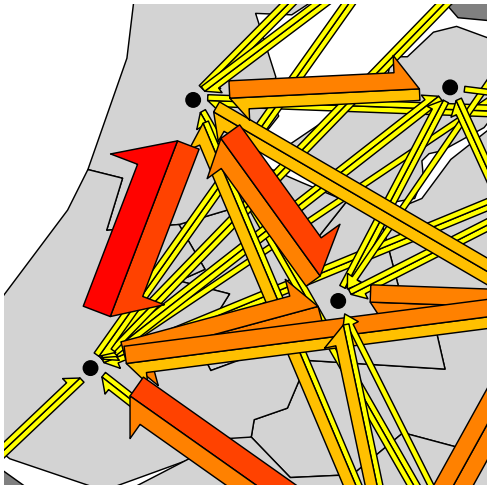
(c) Same as Figure 2.4(b), but with edge shortening.



(d) Same as Figure 2.4(b), but with dynamic edge shortening.



(e) Same as Figure 2.4(d), and with inclusion of raising edge heads.



(f) Same as Figure 2.4(e), and with ordering around vertices (e.g. see the lower left vertex)

Figure 2.4: A sample of a flow map of internal migration of people in the Netherlands, using different methods of edge visualization (data acquired from [5]).

is worth investigating this in the future. References to related material in drawing edges of arbitrary width nicely around obstacles can be found in [12] by Duncan et al..

2.6 Vertex positioning

Initially, one would like to situate each of the n vertices near the center of their respective regions. This can be done by placing each vertex in the center of the *largest inscribed circle* for each polygon. This position may not be ideal at all with respect to certain aesthetic criteria. For each region b_i and its associated vertex v_i we need to optimize the vertex location such that the following criteria are optimized.

Vertex-edge distance. For graphs in general it holds that a proper view the reader should be able to distinguish the vertices and the edges clearly. Hence the distance between a vertex v_i in the graph should not be too close to any of the edges that are not incident to v_i. To comply with this the vertex-edge distance needs to be maximized, but at the same time each vertex needs to remain inside its region.

Overlap with critical features. One of the edges associated with v_i, say $\{v_i, v_j\}$, runs over a critical feature cf. If such a feature is a point then slightly moving v_i and v_j such that the distance between the edge and the point becomes greater than a small constant is sufficient. If the feature is a region the problem becomes harder moving both away might not solve the situation completely. The reason for this is because of the region bounds both vertices are tied to. One can try to minimize the overlap in these cases at best.

Angular resolution. If two edges converge at a vertex with only a small angle between them they have so much overlap that it becomes hard to tell them apart. This situation can be improved by moving either v_i or either of the other two vertices which are at the other end of those edges. Therefore we wish to optimize the smallest angle in the graph, which is by definition optimizing the *angular resolution* of the graph.

Edge crossings. A graph drawing with fewer crossings is in general a better drawing, so another task is to minimize the number of edge crossings. This, however, is difficult to achieve in the given setting as disjoint regions often disallow any movement which minimizes the number of edge crossings. One thing we do wish to take into account is that any edges near a vertex lead to bad situations, because any edge crossings in this vicinity can cause trouble with the overlap of the arrowheads. Therefore the distance between the vertices and the edges should be optimized which coincides with the *vertex-edge distance* criteria.

All of these criteria imply that the vertices need to be moved around. However since these vertices are associated with the regions of the base map their movement is limited to the boundaries of those regions at most. This is strengthened further as positioning the vertex on the boundary itself may lead to undesirable results (e.g. edges ending near the region boundary can confuse the view to which region the edge is pointing). Several types of regions can be used such as circles, squares or a simple polygon that is contained in the region of the base map itself.

2.7 Overview

We've shown several edge visualization techniques such as dynamic edge shortening and ordering of edges to improve the visualization of the data. Next to that the criteria for the optimization of

the vertex positions have been defined. Now we need to analyze the complexity of our problem in order to find out whether an exact solution is possible in polynomial time. As Chapter 3 shows the optimization of vertex-edge distance for all vertices in 2 dimensions is *NP*-hard. Unless $P = NP$ no solution in polynomial running time exists, therefore Chapter 4 is devoted to finding an algorithm for our optimization problem which is based on heuristics rather than an exact algorithm.

Chapter 3

Complexity analysis

In this chapter we show several complexity bounds for optimizing vertex-edge distances in different dimensions. For the required two- dimensional type the problem is already *NP*-hard as proven in Section 3.5. This justifies our choice for an iterative algorithm as treated in Chapter 4. We will first give a formal description of our problem for the optimization of vertex-edge distance with respect to one vertex. After that we go through the one- and two-dimensional settings. The two-dimensional bound can extended to higher dimensions yielding a general bound for all dimensions. After that we reformulate the problem for optimization of all vertices in the graph and show a polynomial bound for the one-dimensional setting as well as an hardness proof for any higher dimension.

3.1 Formalization

Given a graph $G = (V, E)$ with n vertices V and m edges $E \subset \binom{V}{2}$ (i.e. each pair of vertices). Each vertex v_i is associated with a simple region b_i in the plane where all b_i are disjoint. We wish to find the optimal location for each v_i inside b_i such that the vertex-edge distance of a straight line drawing of G is maximized. An example of such an optimization is shown in Figure 3.1. For the optimization the shortest vertex-edge distance for a vertex v_i and an arbitrary edge

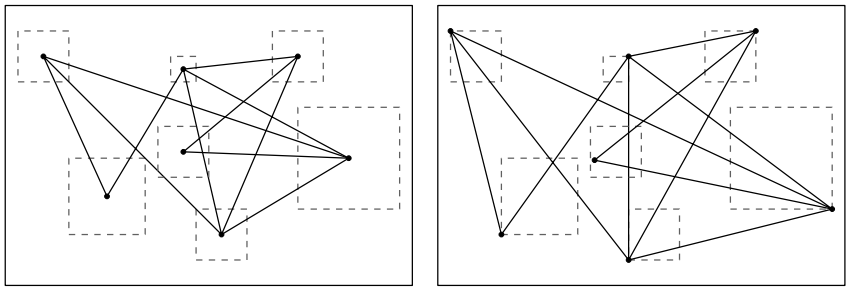
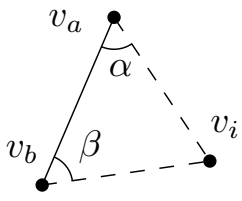


Figure 3.1: Example of vertex-edge distance optimization with square regions.

$e_j = \{v_a, v_b\}$ under the Euclidean metric, also called $\mathcal{L}_2$-metric, is defined as

$$d_j(v_i) = \begin{cases} d(v_i, v_a) & \alpha \geq \frac{\pi}{2} \\ d(v_i, v_b) & \beta \geq \frac{\pi}{2} \\ d(v_i, v_b) * \sin(\beta) & \alpha, \beta \leq \frac{\pi}{2} \end{cases} \tag{3.1}$$



in which $d(v, v') = \sqrt{(v_x - v'_x)^2 + (v_y - v'_y)^2}$ is the Euclidean distance between two vertices v and v'. Given that $d(v_i, v_a) = d(v_i, v_b) \cdot \sin(\beta)/\sin(\alpha)$ it is possible to simplify the case distinction to a single formula of the shape

$$d_j(v_i) = d(v_i, v_b) \cdot \sin(\min(\beta, \frac{\pi}{2}))/\sin(\max(\alpha, \frac{\pi}{2})), \qquad \alpha \neq \pi.$$

If $\alpha = \pi$ then β is equal to zero (the three vertices are collinear). In this case one could switch vertices v_a and v_b to properly calculate the distance. Although the latter is shorter and somewhat easier to implement it is also harder to understand the shape of the curve the function creates. Formula 3.1 shows more intuitively that function has a linear segment in between two parabolic curves.

3.2 Lower envelopes

Figure 3.2 is an example with three edges e_1, e_2 and e_3 along with a horizontal line segment representing a simple region for b_i. Above it is a plot for all three distance functions $d_j(z)$ with $z \in b_i$. We are interested in the minimum over d_1, d_2 and d_3 at every point $z \in b_i$. This minimum is also called the *lower envelope* of $\{d_1, d_2, d_3\}$. Its value at any given point is the function with the smallest value from the given set. This kind of function is denoted in our vertex-edge distance problem as $VE_{\mathcal{D}}$ where $\mathcal{D}$ is the set of distance functions of which the lower envelope is computed.

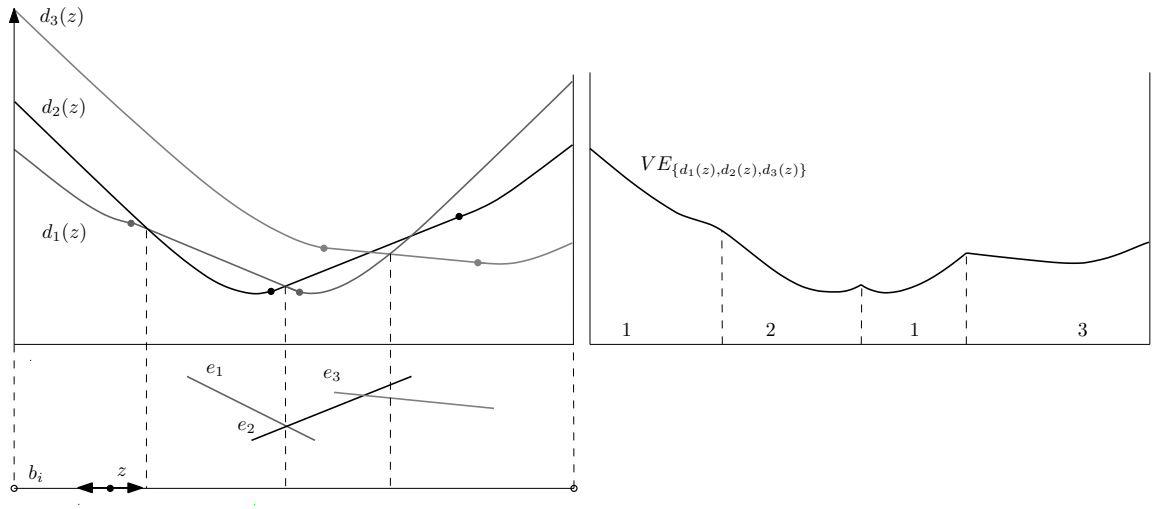


Figure 3.2: Example of a plot for the shortest vertex-edge distance function for three edges with on the right its lower envelope given region b_i as a straight line segment.

We are interested only in the vertex-edge distance of the m' edges in $E' = E \backslash \{v_i, v_j\}$ (i.e. all edges not incident to v_i). The current setting gives the following set of functions $\mathcal{D} = \{d_j(z)\}$, with $j \neq i$. The lower envelope of $\mathcal{D}$ can now be defined as $VE_{\mathcal{D}} : \mathbb{R}^2 \to \mathbb{R}$ that gives the distance between a vertex and the closest edge

$$VE_{\mathcal{D}}(z) = \min_{j \neq i} d_j(z), \qquad z \in b_i.$$

Finding the optimal position v_i^* for vertex v_i is the same as finding the instance z for which the distance function is maximal

$$v_i^* = \max_{z \in b_i} VE_{\mathcal{D}}(z).$$

Once $VE_{\mathcal{D}}$ is computed the optimal position v_i^* can only be found at any of the intersection points between any pair of distance functions on the lower envelope (including the begin- and endpoints of the interval to which the distance function is applied to inside region b_i). These transition points are the only candidates, because it holds for those points that the distance to the closest edges in the graph is locally optimal.

Theorem 1 *The optimal position v_i^* can only be found on one of the intersection points on the lower envelope $VE_{\mathcal{D}}$.*

Proof. Suppose for a proof by contradiction that v^* is not on a transition point, but on an edge of the lower envelope (i.e. a local optimum is generated by a single function). This can never be the case as a curve generated by the Euclidean distance function can only yield a local minimum. The location directly to the left or right of v^* will always improve the metric.

In the degenerate case when an edge is running parallel along the region all values for the distance function are the same. When such a situation occurs one can use the transition points at the beginning or end of that segment to overcome this issue. Finding the best position amongst all those points yields the global optimum by construction of $VE_{\mathcal{D}}$. Hence in order to obtain the optimal position the lower envelope has to be computed and one has to search for the most optimal transition point on $VE_{\mathcal{D}}$ itself. □

3.3 Davenport-Schinzel sequences

The question now remains to determine how hard it is to compute the lower envelope for the given set of distance functions and determining the maximum amount of intersections on it. In other words the bounds for the *algorithmic* and *combinatorial* complexity of $VE_{\mathcal{D}}$ must be proven. Equation 3.1 and the corresponding plots from Figure 3.2 show that the distance function consists of three segments, one linear and two parabolas of second order. A consequence of this is that each pair of functions can intersect at most a constant number of times which will be shown later on. Given these restrictions it is possible to analyze both the combinatorial and the algorithmic complexity of the lower envelope using *Davenport-Schinzel* (DS)-sequences as described in [27]. A DS*(n,s)*-sequence is a sequence $U = \langle u_1, ..., u_m \rangle$ of integers with n being the number of symbols in U and s is defined as the *order* of U. All *DS(n,s)*-sequences satisfy the following three conditions:

1. $1 \leq u_i \leq n$ for each i
2. $u_i \neq u_{i+1}$ for each $i < m$
3. There do not exist $s+2$ indices $1 \leq i_1 < i_2 < ... < i_{s+2} \leq m$ such that $u_{i_1} = u_{i_3} = ... = a$, $u_{i_2} = u_{i_4} = ... = b$ and $a \neq b$.

DS-sequences do not contain any pair of adjacent elements which have the same value and they have no subsequences of length $s+2$ between two alternating symbols. The order of U defines the maximal possible length $|U|$ of a DS*(n,s)*-sequence to be

$$\lambda_s(n) = \max\{|U| \ : \ U \text{ is a } DS(n,s)\text{-sequence}\}.$$

Using this definition and reconsidering the example of Figure 3.2 the DS-sequence for the three distance functions becomes $U(d_1(z), d_2(z), d_3(z)) = \langle 1, 2, 1, 3 \rangle$.

The number of symbols n corresponds to the number of input functions, which equals m' for our problem. For the order s the longest sequence of two alternating functions has to be found. To elaborate the concept of the maximum complexity we give a small example. Consider a pair of straight line segments e_1 and e_2 and use a single variable distance function $f_j(x)$ which

calculates the distance on the y-axis between the edge and x-axis at that point. If the distance is undefined for a value of x the distance can be taken as ∞. The two edges can intersect in at most one location, but because the edges can also intersect with the x-axis the distance function generates symmetrical v-shaped line segments. Therefore $f_1(x)$ and $f_2(x)$ can intersect in at most two locations. This results in the longest possible sequence for $U(f_1(x), f_2(x))$ being $\langle 1, 2, 1, 2, 1 \rangle$ as shown in Figure 3.3. This yields $s = 4$ as there exists no alternating sequence

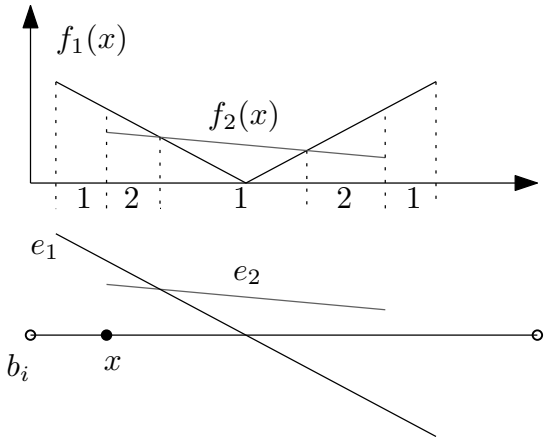


Figure 3.3: Example of maximum complexity for a pair of straight line segments for distance function $f(x) = \min f_1(x), f_2(x)$.

between a pair of functions for a length of $s + 2 = 6$ indices. Given that the distance function $f_j(x)$ is *univariate* (i.e. depends on a single variable) Theorems 3.12 and 3.16 by Sharir and Agarwal [27] yield that $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$, where $\alpha(n)$ is the slowly-growing functional inverse of *Ackermann's function*. Ackermann's function grows so rapidly that its functional inverse is nearly a constant factor for any conceivable input set. This means that even with the exponential factor in the term the function grows slightly slower than linear.

The algorithmic complexity also depends on $\lambda_s(n)$, hence the problem can be solved efficiently using a *sweep-line* algorithm as to be discussed in the paragraph for the two-dimensional vertices with one-dimensional regions in Section 3.4.

3.4 Vertex-edge distance optimization of a single vertex

We first consider our problem for a single vertex in the one-dimensional case and then in two dimensions. In the latter setting we can also make a distinction for the complexity between 1D and 2D regions. For all cases an algorithm can be derived for locating the optimal vertex position within its region in polynomial time.

1D vertices and 1D regions. With the vertices in $\mathbb{R}$ the regions are also effectively reduced to one dimension. Without loss of generality all vertices can be aligned with the x-axis. Optimizing the distance for a single vertex v_i is the same as finding the closest vertex (which is part of an edge) to the left and right and placing v_i in the center. If there happens to be an edge e_j intersecting region b_i then there exists no optimization as it always holds that the distance $d_j(v_i) = 0$, because it was assumed that all regions are disjoint.

For the complexity we need to find the closest neighbors to the left and right of v_i and check whether there is no overlap with any of the edges. Finding these neighbors clearly takes $O(n)$ with n being the number of vertices.

2D vertices and 1D regions. In the two-dimensional version of the problem with one-dimensional regions b_i it can be observed that these become straight line segments which makes them dependant on only the x or the y variable. Without loss of generality we choose to fix the y-coordinate, thus the regions b_i become straight line segments on the x-axis. As a consequence the distance function $d_j(v)$ becomes univariate. This leads to the same type of two-dimensional lower envelopes as shown earlier in Figure 3.2. The value for s is the same as with the vertical distance problem treated in the previous section, because under the Euclidean metric for this problem no new transition points are introduced (see Figure 3.4). With $s = 4$ the combinatorial complexity is also the same, namely $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$.

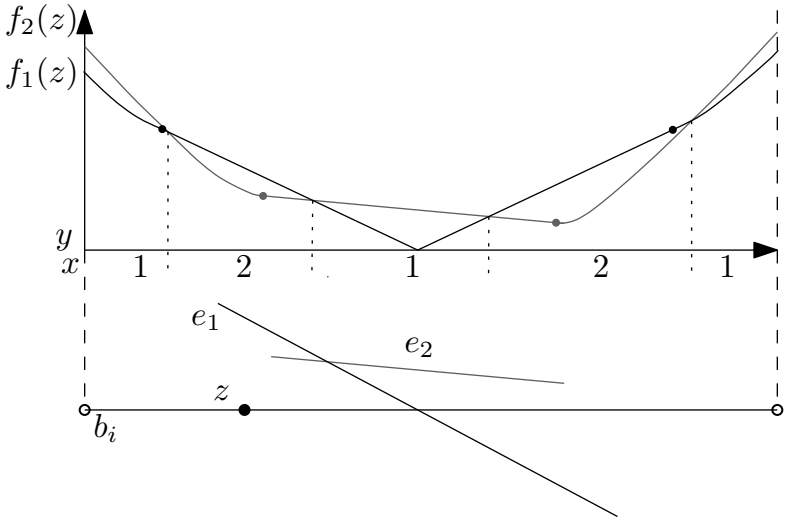


Figure 3.4: Example of a maximum complexity for two edges in 2D Vertex-Edge distance setting with 1D region b_i.

For the algorithmic complexity an efficient solution to this problem is applying a divide-and-conquer strategy on the set of lower envelopes as described in general by Sharir and Agarwal [27]. Let $\mathcal{D}$ be the set of n distance functions. Split $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$ such that each set has at most $\lceil n/2 \rceil$ functions. Compute the lower envelope of both recursively and merge them to obtain the lower envelope for the entire set of $\mathcal{D}$.

Store the transition points between the functions in $VE_{\mathcal{D}_1}$ and $VE_{\mathcal{D}_2}$ in sorted order from left to right in their own respective lists. Next to that also store for each transition point the curve which appears on its lower envelope immediately to the right of the transition, hence include an additional (virtual) point $x = -\infty$ at the beginning. Because each of $VE_{\mathcal{D}_1}$ and $VE_{\mathcal{D}_2}$ can only have $\lceil n/2 \rceil$ curves the length of each list is at most $O(\lambda_4(\lceil n/2 \rceil) + 1)$.

Given the transition points and their associated curves for both $VE_{\mathcal{D}_1}$ and $VE_{\mathcal{D}_2}$ we can use a *sweep-line* from left to right to compute $VE_{\mathcal{D}}$ for the merger step. Starting at $-\infty$ at any point t the sweep-line intersects with exactly one curve $\delta_1 \in VE_{\mathcal{D}_1}$ and $\delta_2 \in VE_{\mathcal{D}_2}$. Let σ_1 and σ_2 be the following transition points in the respective lists, and let ξ be first the intersection point to the right of δ_1 and δ_2 if existent, otherwise let $\xi = \infty$.

The sweep-line moves to point t' which is the leftmost point of ξ, σ_1 and σ_2. Each of the three causes one of the following events to take place. If $t' = \xi$ then we add ξ to the list of transition points of $VE_{\mathcal{D}}$ and associate that point with the either δ_1 or δ_2 depending on which one is lower immediately to the right of it. In the case $t' = \sigma_1$ and δ_1 is below δ_2 then add σ_1 to $VE_{\mathcal{D}}$ and associate δ_1 with that point. On the account of symmetry similar events occur for $t' = \sigma_2$. A depiction of the sweep-line algorithm in action can be seen in Figure 3.5.

In all the sweep-line generates the lower envelope in a number of steps which equals the transition points in $VE_{\mathcal{D}_1}$ and $VE_{\mathcal{D}_2}$ plus the amount of intersection points between the two

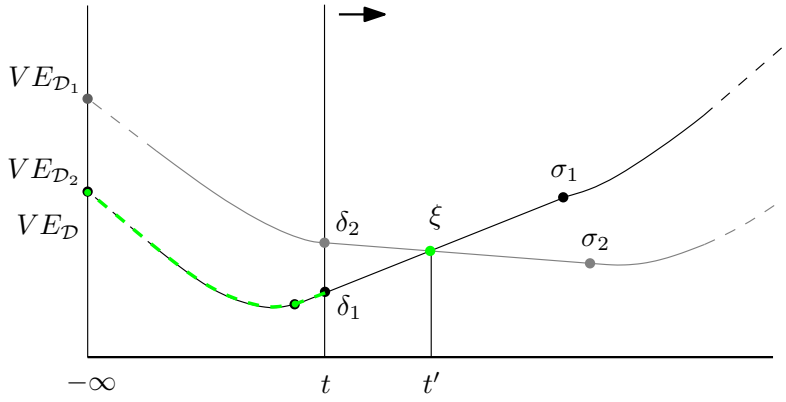


Figure 3.5: Sweep-line algorithm used in the merge procedure for computing the lower envelope of two sets of distance functions. For this instance the next upcoming event is the intersection point at ξ.

envelopes. By construction each intersection point of the two envelopes is a transition point in $VE_{\mathcal{D}}$ and given that we spend $O(1)$ in all the steps the merge requires $O(\lambda_4(n))$ time in total. Let $T(n)$ be the maximum running time for our set of n curves, then the recurrence is:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(\lambda_4(n)) & \text{if } n > 1 \end{cases} \tag{3.2}$$

Using Equation 3.2 the total time then becomes $O(\lambda_4(n) \log n)$. Given that $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$ the $VE_{\mathcal{D}}$ can be computed in $O(n \cdot 2^{\alpha(n)} \log n)$ time. Reviewing the combinatorial complexity shows that $O(\lambda_4(\lceil n/2 \rceil)+1) = O(\lambda_4(n))$, hence putting this all together we can conclude that:

Theorem 2 *Given a set of m edges the optimal vertex-edge distance placement v_i^* for a vertex v_i confined to a straight line segment b_i can be computed in $O(m \cdot 2^{\alpha(m)} \log m)$ time by the sweep-line algorithm as presented above.*

2D vertices and 2D regions. The bivariate function in the 2D setting generates two-dimensional lower envelopes embedded in 3-space. Although the setting is similar it does complicate matters. Since the regions b_i are disjoint any edges in the graph can only intersect b_i, but they cannot start or end in it (apart from the edges connected to v_i itself). Therefore we could try to simplify the problem to a set of piecewise-linear functions which yield a set of n 2-dimensional simplices (i.e. triangles). This allows an easier way to compute the lower envelope using a randomized algorithm. Unfortunately, it turns out that this will not work because there may be situations where a vertex v_j is the closest point of an edge outside the region b_i. A simple counter example is depicted in Figure 3.6 where the vertex-edge distance between v_i and $e_{\{v_j,v_k\}}$ results in vertex

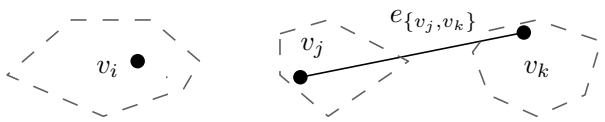


Figure 3.6: Counter example to the argument of using straight line segments in the 2 dimensional setting. The closest point between vertex v_i and edge $e_{\{v_j,v_k\}}$ is vertex v_j itself.

v_j being the closest point. As a consequence of this, the disjointness of the regions does not makes a difference for the complexity.

Because of this we can only use the less tight solution for the general setting. For the analysis of a bivariate *DS(n,s)*-sequence our distance function has to satisfy four criteria:

1. Each function d_i is a set of the form $P(x_1, ..., x_d) = 0$ for some polynomial P of degree at most b. In other words, each d_i is part of an algebraic surface which has a constant degree of at most b (e.g. a cone $x^2 + y^2 = z^2$ would be $P(x^2, y^2, -z^2) = 0$ yielding $b = 2$).

2. The vertical projection on the xy-plane of each distance function d_j is a planar region consisting of algebraic arcs of degree at most b.

3. Given a triple of the given surfaces their relative interiors can intersect in at most s points.

4. The functions in D are in general position, that is the given boundaries of the functions can be defined as polynomials with a constant maximum degree and the coefficients of these polynomials are algebraically independent over the rationals.

The requirements essentially state that the function $d_j(v_i)$ should have a relative simple shape which our Euclidean distance function complies to. To simplify matters we can take the square of the distance function to eliminate the square root from Equation 3.1. A plot generated by the distance function has the shape of a cone which is split in two vertically through the center. In between these two halves the shape is linear. In all the polynomial $P(x^2, y^2, d_j(\{x, y\})^2) = 0$ has two as the highest degree. The last requirement about the general position is only used to simplify the proofs as Sharir and Agarwal show that it involves no loss in generality [27]. This because it has no influence on slightly larger asymptotic bounds for the results that they give. Furthermore by requirement 4 and Bezout's theorem [16] it always holds that $s \leq b^3$, which means that $s \leq 8$ for the vertex-edge distance problem.

Theorem 3 *Given a set m edges the optimal vertex-edge distance placement v_i^* for a vertex v_i confined to simple region b_i in the plane can be computed in $O(m^{2+\varepsilon})$ time, for any $\varepsilon > 0$, where ε is a constant factor depending on s and b.*

Proof. In order to prove this we must show the upper bounds for both the combinatorial and algorithmic complexity of the $VE_{\mathcal{D}}$. Given that our set $\mathcal{D}$ of distance functions comply to the four requirements above Sharir and Agarwal proved in [27] that the combinatorial complexity $\kappa(\mathcal{D})$ of the lower envelope of a set of m bivariate functions $\mathcal{D}$ is $O(m^{2+\varepsilon})$, for any $\varepsilon > 0$, where ε is a constant factor depending on ε, s and b.

Furthermore they also provided an algorithm based on a sweep-line (by e.g. Preparata and Shamos [23]) which can compute $VE_{\mathcal{D}}$ in $O(m^{2+\varepsilon})$ time, for any $\varepsilon > 0$, where ε is a constant factor depending on ε, s and b, under the same assumption that the bivariate functions in $\mathcal{D}$ comply to the four requirements as mentioned above. Hence the total running time required to find the optimal position on the lower envelope is $O(m^{2+\varepsilon})$. □

3.5 Vertex-edge distance optimization of n vertices

Now that we have determined the complexity of optimizing a single vertex we look at the optimization of the minimal vertex-edge distance of all n vertices. To this end redefine function $VE_{\mathcal{D}} : (\mathbb{R}^2)^n \to \mathbb{R}$ in the same way as before

$$VE_{\mathcal{D}}(z_1, ..., z_n) = \min_{1 \leq j < m'} d_j(z_i), \qquad z_i \in b_i.$$

Finding the optimal vertex set V^* for which the smallest distance between any edge and vertex pair is optimal becomes the same as finding instances z_i for which the distance function is maximal

$$V^* = \max_{(z_1,...,z_n)\in b_1\times...\times b_n} VE_{\mathcal{D}}(z_1, ..., z_n).$$

The complexity of the problem increases rather quick leading to *NP*-hard results for the 2D setting as shown in the following paragraphs.

1D Vertices and 1D regions. The optimization for the one-dimensional setting is essentially the same as optimizing vertex-vertex distance. This problem in one dimension leads to a family $\mathcal{D}$ of intervals on real numbers. In such a situation we can formulate the problem as a scheduling problem: Let V be the set of tasks to be executed without preemption on a single processor. All tasks are processed with same time length q which are to be scheduled with the given release and due times respectively r_i and d_i. With the sets $D_i = (r_i + \frac{q}{2}, d_i - \frac{q}{2})$ on the family $\mathcal{D}$ the question is whether a feasible schedule exists. The processor scheduling algorithm mentioned in [4, 14] and described by Simons [28] which runs in $O(n^2 \log n)$ provides a solution to this problem.

For vertex-edge distance the problem becomes more complex as the number of vertex pairs is an order of magnitude smaller than the number of vertex-edge pairs. Because all vertices are one-dimensional and their respective regions are disjoint, the vertex-edge distance metric simplifies to the vertex-vertex distance metric. Disjointness also means that the order of the vertices can never change no matter how the vertices move. If a vertex v_i is connected by an edge with another vertex v_j, then v_j doesn't influence v_i's schedule but the neighbor of v_j does. All regions are given as disjoint, therefore a vertex can only have a degree of at most two. If we store for each vertex its two neighboring vertices this allows us to find in constant time the closest non-connected vertex which does influence the schedule of v_i. Because the order of the vertices never changes this introduces no additional complications for the computation of the schedule.

Theorem 4 *The optimal vertex-edge distance placement for n vertices and m edges in $\mathbb{R}$ where each vertex v_i is confined to a region b_i with all regions b_i disjoint can be computed (if possible) in $O(n^2 \log n)$ time.*

2D vertices and 1D regions. Currently this is still an open problem. The algorithm for the one-dimensional version cannot be generalized to two-dimensional vertices, because an arbitrary number of processes can be running simultaneously. An *NP*-hardness proof has not been found either as we cannot reuse the arguments for the two-dimensional regions as shown below.

2D vertices and 2D regions. Again considering the same situation as for the single vertex optimization we now prove that optimizing the Vertex-Edge distance problem for n vertices in 2D is *NP*-hard. The associated decision problem for this optimization problem is

$$\max_{(z_1,...,z_n)\in b_1\times...\times b_n} VE_{\mathcal{D}}(z_1, ..., z_n) \geq t,$$

in which t is an arbitrary value greater than 0.

Theorem 5 *The decision version of the vertex-edge distance placement problem for m edges in $\mathbb{R}^2$ where each vertex v_i is confined to a simple region b_i in the plane is* NP*-hard.*

Proof. To show *NP*-hardness for the decision problem we have to show that it can be reduced from another problem which is *NP*-hard.

The decision problem of vertex-edge distance placement can be restricted to the decision problem for *Spreading Points* which was shown *NP*-hard by Cabello in [4] using reduction from the distant representatives problem as shown by Fíala et al. [14]. Spreading Points problem optimizes vertex-vertex distance in which the vertices can move around in (possible intersecting) disc shaped regions. Our problem uses simple regions which allow restriction to discs. Furthermore consider that by taking all edge distances as zero all edges collapse into single vertices for which the distance function becomes the same as the Euclidean metric used for Spreading Points. We have now restricted our problem to Spreading Points which shows that vertex-edge distance optimization for 2D vertices and regions is *NP*-hard. □

Now, because the optimization problem is at least as difficult as its associated decision problem it also holds that the optimization problem is *NP*-hard.

3.6 Overview

The proof for *NP*-hardness can be extended generically to higher dimensions. For the single vertex $v_i \in \mathbb{R}^3$ the combinatorial complexity holds for any higher dimension $O(m^{d+\varepsilon})$ with d for the number of dimensions. The algorithm for calculating the lower envelope can only be extended to trivariate functions using a randomized algorithm running in $O(m^{3+\varepsilon})$ [27].

This algorithm cannot be used for any higher dimensions leaving the problem open for $\mathbb{R}^d$ with $d \geq 4$. Agarwal et al. [2] show that the vertices of the lower envelope in d dimensions can be computed in $O(m^{d+\varepsilon})$. Since we only need to look at the vertices of the lower envelope to find the optimal position this will do. All results are shown in Table 3.1. The complexity

$v_i \in \mathbb{R}^d$	$b_i \in \mathbb{R}^d$	Single vertex	n vertices
$d = 1$	$d = 1$	$O(n)$	$O(n^2 \log n)$
$d = 2$	$d = 1$	$O(m * 2^{\alpha(m)} \log m)$	?
$d \geq 2$	$d \geq 2$	$O(m^{d+\varepsilon})$	*NP*-hard

Table 3.1: Overview of complexity for various settings for vertex-edge distance optimization

bounds of the problem for two-dimensional vertices with one-dimensional regions is an open problem at this moment. Another thing to consider is that the restriction of disjoint regions b_i was only required for the 1D vertex-edge problems. We could consider the general case to see what the differences in complexity are when we give up the disjoint region restriction. Lastly, the time-bound of $O(n^2 \log n)$ for optimizing one-dimensional vertices is very likely not optimal, because in this setting the vertices can never change their order.

Chapter 4

Algorithmic approach

4.1 Introduction

Suppose we are given a base map consisting of n simple polygons representing the regions of the map and a corresponding $n \times n$ table with input values for the flows between each pair of regions. All flows are represented by edges between these regions placed on top of the base map. As a design decision we choose to associate each region with a single vertex v_i in such a way that all edges related to that region have v_i as its begin- or endpoint. Alternatively, one could have chosen to allow the endpoints of each edge to float around in the region independently. This yields a graph $G = (V, E)$ with n vertices in V and m edges in $E \subset \binom{V}{2}$ defined by the input table. Each vertex v_i is confined to its own respective regions b_i. A region b_i is defined as a shape which is contained in the input region that is given on the base map. An example of a map with such regions can be seen in Figure 4.1.

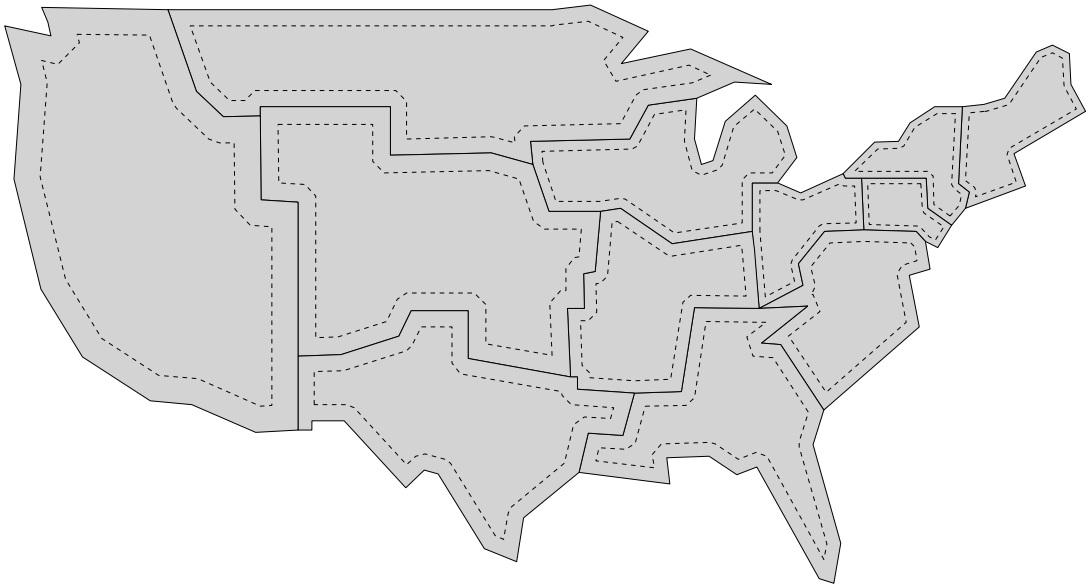


Figure 4.1: Example map showing region boundaries b_i as polygons with dashed edges for each input region.

We wish to find the optimal location for all vertices within their corresponding regions such that the resulting flow map is aesthetically pleasing conform the criteria as treated in Chapter 2. In Chapter 3 we showed that even if we restrict ourself to optimizing vertex-edge distance in two dimensions the problem is already *NP*-hard. Unless *P*=*NP* it is not possible to find an optimization which runs in polynomial time. Including any of the other criteria will naturally only make this problem harder. Instead of calculating the exact optimal placement we have to

adjust our goals and look for a generally good solution with an acceptable running time given the restrictions and criteria we must comply to.

All vertices are confined to their own respective regions implying that optimization can only occur at a local level. This suggests a *neighborhood search* as method of optimization. Such an optimization technique repeatedly optimizes the current solution using a predetermined set of local operators. Starting out with a initial configuration the algorithm iteratively repeats the optimization, until no further improvement can be made. The most common problem however with this type of optimization is that the solution can get stuck in a so-called *local minimum*. Such a situation prevents any further optimization as all neighboring solution are worse than the current solution. In our case this could occur in areas that are crowded with edges. Since a flow map should be reasonably sparse in order to be readable to the viewer we assume that our input set does not result in the worst-case. Next to that we apply a *cooling schedule* as defined later on in Section 4.3.1 which allows one to overcome problems with local minima. From this it can be concluded that one can try to implement an algorithm based on sensible "rules of thumb" for each of the criteria.

4.2 Related research

Algorithms that optimize aesthetic criteria for graph lay-outs use different types of strategies such as spectral, orthogonal, force-directed, tree and hierarchical lay-outs to name a few. Our problem is most similar to those often tackled by force-directed graph-drawing methods.

As explained in Section 1.4 force-directed methods calculate the forces exerted between the vertices and the edges based on the analogy with physical systems made up of rings and springs. The attractive and repulsive forces caused by the springs move the vertices around until a state has been reached where the total amount of force is reduced to a minimum. Depending on the applied criteria these forces can influence the vertices' movement in the plane. The most common aesthetic criteria applied in this setting are optimizations regarding vertex-vertex distance, edge crossings, uniform edge length, reflection of symmetry and angular resolution. Several papers have been written about this topic and each of them uses a different approach to optimize the graph as much as possible depending on the criteria the authors of the respective papers find most valuable.

All force-directed algorithms follow more or less the same approach. They start out with a given initial configuration. This configuration can be set up to prevent any bad situations from occurring at the beginning which could get the algorithm stuck. For most algorithms a random placement works fine as long as it prevents overlap of two vertices. It then performs three basic steps during each iteration: computation of the effect of both attractive/repulsive forces and delimiting maximal displacement to prevent any unbound repulsions. Lastly, when a termination requirement has been met, the iterative process comes to a halt. This requirement is not easy to define in numbers as test results by all of the methods below show that no specific constant or a value depending on the number of vertices/edges will suffice in all situations. Reconsidering the analogy with the physical system the optimization of a graph continues, until the total amount of energy in the graph reaches a state where it is minimal.

Eades [13] was the first to use the concept of a spring embedder to optimize the lay-out of a graph. He introduced the main concept on which all of the succeeding papers are based upon. Eades, however, did not implement Hooke's law as shown in Equation 1.2 which reflects the physical analogy, but he choose his own formula to calculate the exerted forces placed upon the graph. Next to that he reasoned that a vertex only needs to be close to its immediate neighbors. This means that only the attractive forces between neighboring vertices need to be calculated, which can be done in $\Theta(|E|)$ time. All repulsive forces still need to be computed in between all vertex pairs. For the termination requirement Eades used a constant value of 100 iterations

because that worked out pretty well for most graphs.

Kamada and Kawai [17] modeled the physical forces after Eades, but they solved the partial differential equations created by Hooke's law. Their processes stays close to the original concept from the physical model by reducing the sum of compression and tension on all the springs to reach an equilibrium state. For the termination requirement they check each iteration whether the total amount of energy drops down below a preset value. Because their algorithm moves only one vertex at a time only the forces with respect to that vertex need to be computed which takes $\Theta(|V|)$ time for the inner loop to complete.

A different method by Davidson and Harel [9] also involves a method of reduction of the total amount of energy in the graph. Their method is based on a powerful, generic, but computationally costly process called *simulated annealing* which is also used in various other optimization problems. They choose an energy function which combines the terms for all criteria (e.g. vertex-edge distance, edge-crossings, etc.). Furthermore their method of delimiting maximal movement is very loose in the beginning and gradually becomes stricter over time. Hence in the beginning large leaps are possible whereas later on more fine grained adjustments can be made. This progressively more tight delimitation of movement is often referred to as the *cooling schedule*. This originates from the analogy of the behavior of particles in the annealing process. The particles have greater mobility under high temperatures and can move only very little if the temperature drops to a lower level. Using this approach graphs with a high quality are attainable, but as with all processes based on simulated annealing it comes at the price of being rather slow. This is caused by the gradually decreasing cooling schedule as bounding vertex movement more rapidly can lead to problems with local minima.

Fruchterman and Reingold [15] base their simulation on only two simple principles to compute new vertex locations. Vertices connected by an edge should be drawn near each other and the vertices themselves should not get too close to each other. Hence two vertices connected by an edge will attract each other, but any pair of vertices at close range repel each other. This can lead to problematic situations if two vertices are very close, because the repelling forces become too large. To overcome this problem they allow the vertices to move in a discrete manner allowing the vertex to move past an edge in a single step. Similar to Davidson and Harel they delimit the maximal displacement by a temperature variable which gradually decreases over time. They also make their algorithm's termination depend on it, because when the temperature gets close to zero only little movement is possible and finishes the optimization process of the graph. Fruchterman and Reingold also provide a variation on their algorithm which has a shorter running time. In order to speed up the algorithm they divide the plane in equally sized grid boxes. By doing so they can compute the repelling forces between the vertices more efficiently as only the vertices of neighboring boxes need to be checked.

Lastly, the method as proposed by Lin and Yen [18] focusses on optimization of angular resolution by edge-edge repulsion. Their approach differs from the other that they do not mind about vertex overlap at all. This leads to graphs which look somewhat different from the graphs generated by the other force-directed methods which do prevent vertex overlap.

4.3 Force-directed approach

Our problem poses an additional restriction of movement on the vertices in the graph as they are confined to the given set of disjoint regions. This implies that our graph is already in a near-optimal situation to begin with and it is only possible to make improvements on a local scale. The force-directed methods described above all had to take into account that the vertices must remain close to each other, because otherwise the graph would explode out of proportion. For our problem such an expansion cannot occur. Now let us consider any of the distance metrics used in the previously mentioned approaches. They all use a function to compute the

repulsion based on the actual distance. Such a function would not yield satisfactory results here as vertex movement is very limited. Taking these matters into account shows that a modification is required for this method to be successful under the given circumstances. Our method closely relates to what Fruchterman and Reingold did, but because we wanted to overcome local minima by allowing jumps outside the current neighborhood it also influenced by the simulated annealing approach by Davidson and Harel.

Reconsidering the general approach we can modify any parts accordingly to meet our needs. For the initial configuration any set of vertices which are already inside the corresponding regions will do. For convenience we place the initial vertices in some center of the given region. The polygons defining the regions are simple so it does require care to place them properly. More on this topic in Section 4.4. Calculation of the attractive and repulsive forces is not the same, as it is not required to keep the vertices close together explicitly. However without any attractive forces all vertices will remain moving endlessly throughout the region, because an equilibrium can never be reached. Therefore we apply a cooling schedule which gradually restricts movement until all vertices come to a halt. A property of such a cooling schedule is that we can use this as our termination requirement as well. Now that we globally outlined our algorithm we can formally describe the cooling schedule, the termination requirement and the calculation of the forces. Next to that we introduction the notion of boundary clipping, because our vertices need to remain inside the given regions.

4.3.1 Cooling schedule

The maximal allowable distance for the movement of vertex v_i is defined as $m_i^{\max}$. For the first iteration $m_i^{\max}$ is computed as the distance between the two points which are farthest away from each other on the region's boundary, because we wish to allow maximal movement in the first few iterations. After the first loop the distance has to decrease each consecutive iteration slowly towards zero. This can be done using a function which is either linear or exponential for example. A linear function however tends to diminish the allowable movement too rapidly causing problems with local minima. The problem with a linear function is that it allows equally many large movements as small movements, whereas a few large jumps in the beginning followed by many smaller optimizations works out better. This can be reflected on the gradually decreasing temperature for the simulated annealing process which following the physical model uses an exponential function [9]. Thus we try to look for a simple function. After several experiments it was found that dividing $m_i^{\max}$ by the number of iterations works out, but the function dies out somewhat too fast. Therefore the formula was slightly modified to:

$$m_i = \frac{m_i^{\max}}{1 + iteration * c},$$

In which m_i is the distance the vertex can move, *iteration* stands for the iteration number starting at 0 and c is a given constant with a domain of $\langle 0, 1 \rangle$. This function results in a few large steps in the beginning and many smaller local movements after that. The steepness of this curve depends on c which we have taken as 0.5. Because this function converges to zero slowly the differences between the mobility of vertices with large and small regions is reasonably small. For a linear function a vertex with a small region in the graph would reach an equilibrium sooner than any of the neighboring vertices that have more freedom. Other cooling schemes which can be used are simple exponential functions such as initializing m_i as $m_i^{\max}$ and decreasing m_i by a constant percentage each iteration.

4.3.2 Region boundary clipping

If either one of the endpoints passes b_i's boundary the edge is clipped off there and that endpoint on the region is taken instead. It does make a difference if one takes the location of the intersection itself or if the vertex is allowed to slide along the boundary towards the location minimizing the Euclidean distance to the unreachable endpoint. For any type of region it seems that allowing the vertex to slide along the boundary generates better results as the vertex tends to get stuck on the intersection point with the boundary otherwise. Figure 4.2 shows the difference between (dis)allowing movement along b_i's boundary for a square shape.

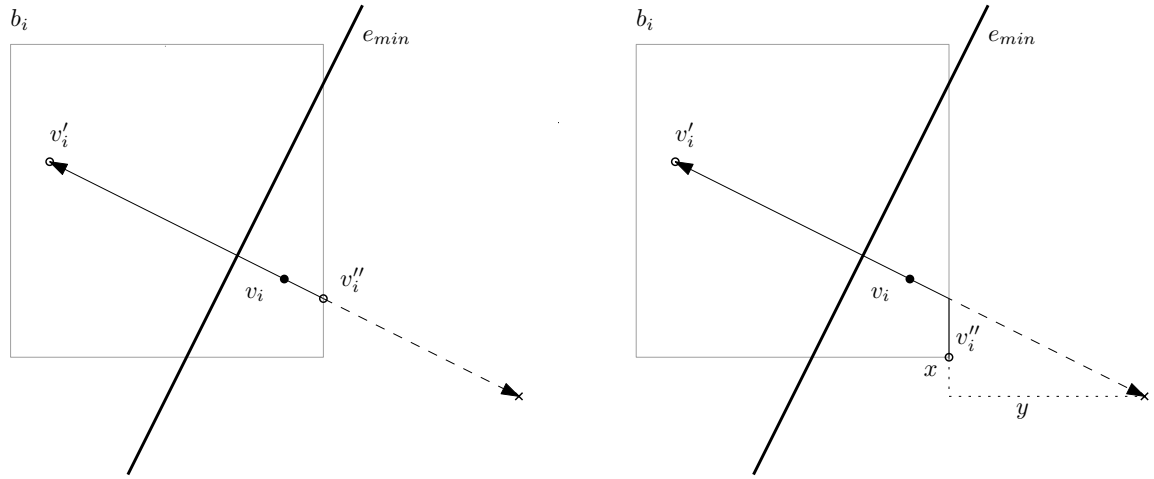


Figure 4.2: Boundary clipping by taking the actual cross point as shown left, or by splitting the vector's x- and y-components and using those to move the vertex to the closest point near the desired location.

4.3.3 Force computation

The most important part of the algorithm is the actual computation of the forces. These allow us to optimize the graph to our own preference. In Chapter 2 four criteria were defined to which the algorithm must comply to namely, vertex-edge distance, angular resolution, avoidance of critical features and edge crossings.

Vertex-edge distance. Optimization of this metric means that the distance for each vertex with respect to the closest needs to be maximized. The formula $d_e(v_i)$ used for this metric of vertex v_i and any of the m' edges in $E' = E \backslash \{v_i, v_*\}$ can be defined as following:

$$d_e(v_i) = \min_{1 \leq k \leq m'} dist(v_i, e_k)$$

With $dist(v, e)$ being the Euclidean distance function between vertex v and edge e. Calculating this distance also allows one to determine the closest edge e_{min} as well. To improve this distance vertex v_i moves along the axis perpendicular to e_{min} in both directions with distance m_i. The endpoints v_i' and v_i'' are generated which are towards and away of e_{min} starting at v_i respectively with length m_i. Because we allow movement in both directions the vertices have to move discretely. Our motivation for doing so is that a vertex can never overcome the local minima when it is about to cross an edge. To decide whether to use v_i' or v_i'' the location with the largest distance to e_{min} is taken.

We could argue that this method yields poor results as we do not take any of the other edges in the graph into account. However very dense graphs do not make up useful flow maps in practice.

Another thing to consider is that due to the movements of the other vertices the graph may look completely different from a local point of view, hence any exact or complex computations for a single vertex will not yield much better results either. But apart from those reasons allowing vertex to move to a worse situation allows the vertices to escape the problematic situation generated by local minima. During the first few iterations a lot of these jumps occur, but for the later iterations this degree of freedom decreases until only local movements are possible permitting final adjustments to the graph.

Angular resolution. The second metric to consider is the angular resolution ar_i of a vertex v_i. This property is defined as the smallest angle between any pair of edges that come in at the same vertex v_i provided that v_i has at least two edges attached to it. In the same context the formula for this metric becomes:

$$ar_i = \min_{\{v_i,v_j\},\{v_i,v_k\}\in E} angle(\{v_i, v_j\}, \{v_i, v_k\})$$

with $i \neq j \neq k$ and $angle(e_v, e_w)$ is the smallest angle between two edges e_v and e_w. This metric could be incorporated into the vertex-edge distance metric updating them both at the same time. Observe that most of the flow maps often have some specific regions which are important (i.e. have many edges connected to them). If the region is not in the center of the map, then all edges will come in at the associated vertex with small angles creating a small angular aperture as shown in Figure 4.3. This cone shape prevents any significant improvements

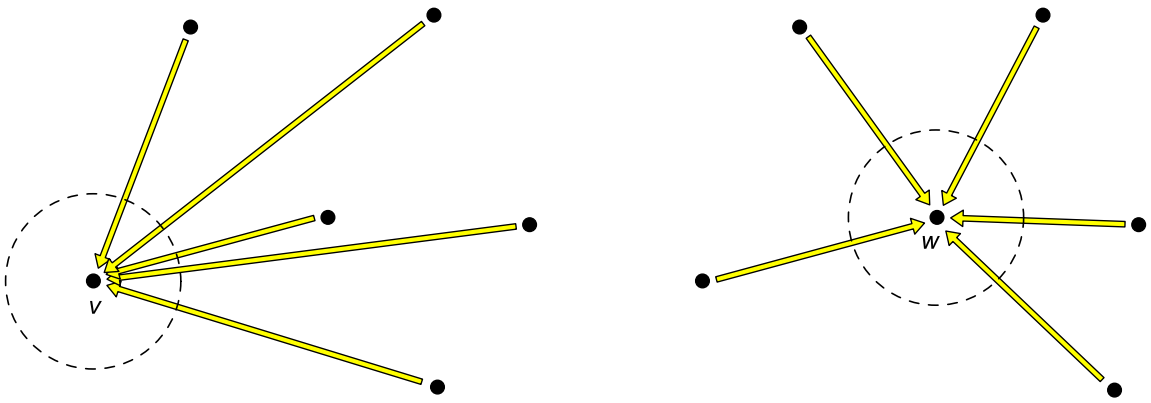


Figure 4.3: Vertex v has all edges coming in from the same side resembling a cone making it hard to optimize angular resolution for this vertex. Vertex w is in the center of its respective graph and therefore it can optimize its position for the same metric much easier.

for angular resolution, because no matter where the vertex is placed inside the region the angles are still relatively small. It would be computationally costly to update the metric each time a vertex is moved during an iteration. Instead we update it once during an iteration after vertex-edge distance has been optimized for that vertex.

Restricting ourselves to local movements the optimization can be confined to an area bounded by the closest edge and the maximal allowable displacement factor m_i. Hence the maximal allowed movement for vertex v_i with respect to angular resolution is $m_i^{ar} = \min(m_i, d_e(v_i) - c_2)$, where $c_2 \geq 0$ is a constant value depending on how close one is willing to let the vertex get to the closest edge. For the implementation of our mapper a value of 20 pixels is sufficient as the arrow heads and tails do not overlap with the closest edge if this distance is taken. After computing the vertices in question and determining for each vertex its allotted movement m_i^{ar} one can compute the optimal location for all three vertices. An example is shown in Figure 4.4.

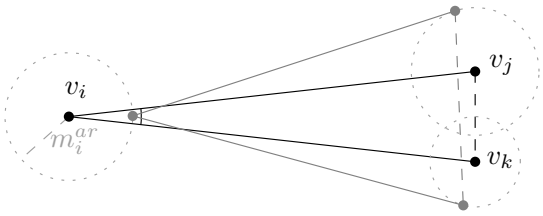


Figure 4.4: Improvement for angular resolution metric.

Avoidance of critical features. The metric for the set critical features CF is harder to define. What can be accounted for as overlap? Does it only count when a feature is covered completely, or if the distance to the feature is less than a certain constant threshold. One can also consider giving up a critical feature when it is covered by several edges, so attention can be focused on the other features (i.e. is seeing all features barely aesthetically better than seeing some with a clear view?). The preference goes to giving up features which are very hard visualize. Reasoning for this is that visualizing every single feature may be impossible given the regions and the fact that the edges often concentrate around certain regions. Even if it can be made visible then there's hardly anything to see of its surrounding environment. This makes it hard to judge its locations, hence one could favor the other approach. For each feature $cf \in CF$ the metrics are defined as the number of times the feature has overlap cf_o and the shortest distance cf_d between the feature and any of the m edges in E:

$$cf_o = \#_{1 \leq k \leq m} dist(cf, e_k) = 0$$

$$cf_d = \min_{1 \leq k \leq m} dist(cf, e_k),$$

where $dist(cf, e)$ is the smallest Euclidean distance between critical feature cf and edge $e = \{v_i, v_j\}$. Optimization of the feature is only useful if the number of overlaps is less than a constant minimum $cf_o \leq c_3$. The value of the constant c_3 is rather small. Supposedly, when three or more edges cover up the feature, then even if all three edges improve the distance to the feature locally, then still only very little can be seen of the feature and its direct surroundings. Hence we can consider optimization of such a feature only worthwhile if this value less than three.

If a feature requiring optimization is found we allow to move one vertex away from the feature per iteration. To that end the vertex with the best ratio between being closest to the feature and having the largest degree of freedom for movement is chosen. Similarly to what happened when updating angular resolution the distance in which either vertex v_i or v_j has to move is $m_c = \min(d_e(v_i), c_4 - d_{ce})$ in which d_{ce} is the distance between the feature cf and edge e, and c_4 is a small constant representing the minimal distance that is needed to make the feature visible clearly.

For our implementation we chose 8 pixels as this distance shows enough of the feature's surroundings to make it recognizable. The angle in which v_i moves is perpendicular the edge between v_i and v_j in order to maximize the effect of the movement. Again, any movement outside the region is clipped off at the region's boundary. Hence we should allow movement in the opposite direction as was used for determining the best location for vertex-edge distance.

It may be the case however that v_i's old location also causes other critical features to get blocked. Therefore the vertex which obstructs the most critical features has to be the first one to move. The distance then becomes the maximum of all the displacements m_c. Since the maximal displacement is chosen the angle should be taken from the corresponding angle for that feature. One could try alternatives such as taking the average, but this can only work if all angles are in the same quadrant as opposing movements negate each other.

Prevent bad edge crossings. Because of the local nature of movement which is allowed by the vertices most improvements for this metric only apply in certain special cases. Hence instead of minimizing the number edge crossings the graph it would be a better approach to minimize edge crossings near the edge heads or the direction of flow would become ambiguous otherwise. This metric however coincides with vertex-edge distance so no additional optimizations need to be applied. Due to the nature of certain maps bad situations can still occur, hence we still need to solve problems regarding edge weaving and arrowhead obfuscation. This is a visualization problem and is discussed in Section 4.5).

4.3.4 Termination requirement

A termination requirement is necessary to stop the iterative process when a good state has been reached. One could take a constant number of iterations, but then there is hardly anything useful to be said about the value of this number. Inspecting the cooling schedule shows that it converges slowly to zero. We can use this to stop the process when all $\max_{1 \leq i \leq n}(m_i) < \varepsilon$ for a given constant ε. The value of ε can be taken as one pixel or slightly larger than that, because after that many iterations all vertices can move only so little that any further improvements are of no significance to the human eye.

4.4 Computation of the region boundaries

A proper definition of the regions b_i has been left open so far. We could take the regions on the base map themselves as a restriction. This however can lead to confusion if the vertices end up on the boundaries of those regions, because the region to which the arrows are pointing will become ambiguous. The regions are also arbitrary simple shapes leading to very unideal optimizations, therefore one could try to work with a simple shape such as the largest inscribed circle, square or convex polygon. For the latter we do not necessarily need a convex shape, because the vertices move around in a discrete manner. Instead looking for a large polygon with a high degree of *fatness* is more interesting, where a polygon's fatness is defined by the ratio between the smallest enclosing circle and the largest inscribed circle of a polygon. For all of these shapes it goes that they are shrunken down by 10% to avoid any possibilities where vertices may end up on the region's boundary.

Largest inscribed circle Let P be a simple polygon in the plane with n vertices p_i. The largest inscribed circle LIC can be defined as the circle of which its center is a point v inside P for which it holds that it is farthest away to all of the edges which P is composed of. If one would move all the edges inwards perpendicular to their own respective axis with the additional requirement that concave vertices turn into curves the last remaining point is v. The edges which are created by the movement of vertices p_i during this shrinking process generate the *medial axis* $M(P)$ of P. This medial axis has the property that the LIC is always on one of the vertices on $M(P)$. An example of this shrinking process, the corresponding medial axis and the LIC of a simple polygon can be seen in Figure 4.5.

Chin et al. [7] show a method based on histograms to compute this medial axis in $O(n)$ time and thus it allows for calculation of the largest inscribed circle in linear time as well. In our case an implementation is required for polygons with a relative small value of vertices ($n < 50$). Because of this a slower, but more simple approach was taken to find the largest inscribed circle which follows the method as described by Preparata [22]. Skipping the details which are explained in the following paragraph general one can say that he computes the medial axis by removing the segments in P one by one, because each event of three segments intersecting at point c generates a vertex on the medial axis. Provided that the associated circle C with c as it

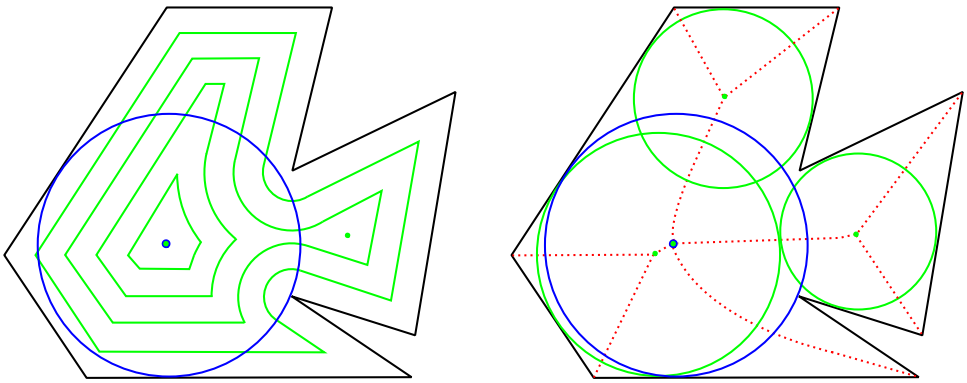


Figure 4.5: The largest inscribed circle of a simple polygon by moving the edges inwards. The figure on the right shows the medial axis as dotted lines generated by the vertices' movement during the shrinking process.

center and radius r being the distance to each of the intersecting segments must lie completely inside P. To find the largest inscribed circle itself we only need to reduce polygon P to a single vertex. The implementation sketched below uses the algorithm as shown in the same paper. The only difference is that any additional steps to construct the medial axis itself are left out.

Given our vertices in sorted order around the polygon's boundary one can store them as a list of segments S. These segments are all the edges and also includes the concave vertices separately which will represent the curves. If we have m concave vertices than the list will have $n + m$ items with $m \leq n - 3$, hence the number of segments is linear with respect to n. We now calculate for each triplet $\langle S_i, S_j, S_k \rangle$ of consecutive segments their intersection point c and store this in a list ordered by the radius of C (if there is no intersection take the radius as ∞). This can be done in $O(n \log n)$ time. We pick the smallest intersection point c from the list and check it against all edges in $O(n)$ time. If there is an edge intersecting the circle at more than one position we continue with the next event, otherwise we remove S_j and recalculate the new values for the two neighboring segments. Storing these new values in the sorted list for each event takes $O(2 \log n)$ time. This needs to be done until there is only one triplet left in the list (i.e. only three segments) for which the last remaining intersection point, which is also the center of LIC, can be found in constant time.

The algorithm requires $O(n)$ time per iteration and there are $O(n)$ iterations at most. This yields a running time of $O(n^2)$, which is fine for the any arbitrary simple polygon with the limited amount of vertices used in the implementation. An example of the final output where all circles have been shrunken down by 10% can be seen in Figure 4.6.

Largest inscribed square From a computational point of view the largest inscribed axis-parallel square LIS can be found using the same method as shown for the circle. The difference is that instead of using the $\mathcal{L}_2$ Euclidean metric we now use the Chebyshev metric also referred to as the $\mathcal{L}_\infty$-metric. For two vertices v and w the metric is defined as $d_\infty(v, w) = \max(|v_x - w_x| + |v_y - w_y|)$.

Largest inscribed polygon Apart from using circles and squares the use of the largest internal convex polygon inside simple polygon P is interesting enough to investigate. Chang and Yap [6] show how to compute this convex polygon. They describe a polynomial algorithm, but unfortunately the algorithm has a running time of $O(n^7)$ due to the difficulty of determining which internal convex polygon has the largest area. Even for our simple polygons with at most 50 vertices it would be computationally too expensive to implement.

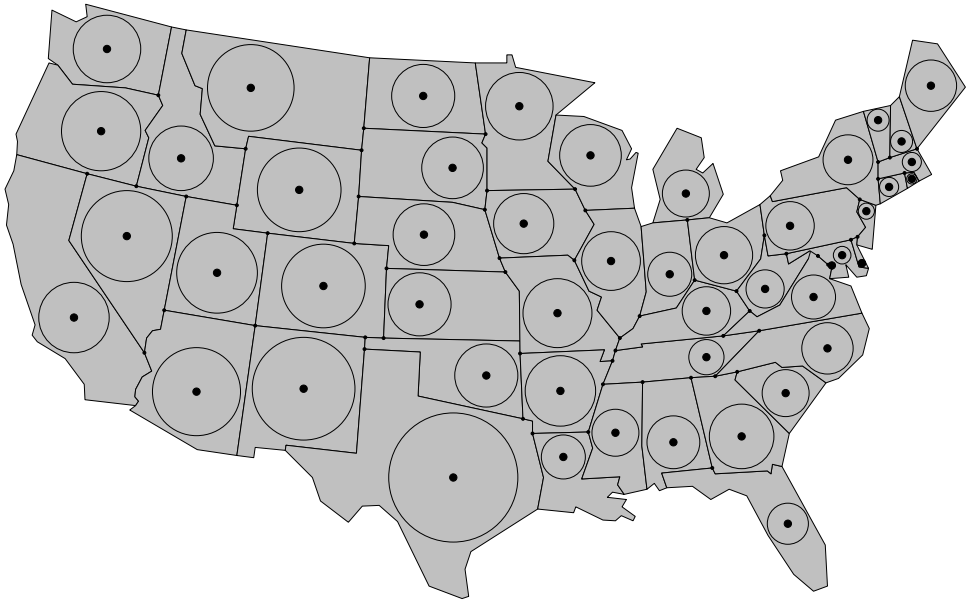


Figure 4.6: The set of largest inscribed circles shrunken down by 10% for all 48 states on the continental USA.

Because our vertices move discretely the use of simple polygons should work out. The boundaries themselves can be taken for the regions, but it is not hard to imagine that certain shapes can cause difficulties for our iterative process. To overcome this problem one could use the notion of *fatness* for a polygon. The diameter of a simple polygon P is the diameter of the smallest enclosing circle and the width can be defined as the diameter of the largest inscribed circle. The ratio α between these two determines the fatness of the polygon, where a small α represents a high degree of fatness. Damian [8] describes an algorithm that decomposes a given simple polygon into set of α-fat polygons in $O(m^4n^3)$ with m being the number of edges in the visibility graph of P. Again, such an algorithm is not very practical to implement.

Therefore we resorted to drawing these polygons automatically for base maps used in our experiments. The given polygons on the base map are shrunken down by the required distance which yields a polygon which is 10% smaller than the original. This is done by moving all edges inwards perpendicular to their own respective axes (see also Section 4.4). Any curve generated by a concave vertex is removed and replaced by a straight edge which connects both endpoints.

4.5 Edge visualization

After the graph has been computed all edge segments are visualized one by one on top of the base map. This can cause problems with some of the edge heads as they can get hidden underneath a bypassing edge. The method as described in as treated in 2.4 can help minimizing this problem, but it can never solve it completely. To make sure that an edge head always appears over the blockading edge one can compute the intersection points the other edges have with this edge. Then find the closest location to the arrowhead where the edge is visible over the entire width. Cut the edge in two at that position to create two open polylines namely the *head* and the *tail*. The tail can been left as it was, but the head needs to be raised in height to appear over the edge which was obfuscating it. An example of this method is shown in Figure 4.7. This works fine for most instance, but if an edge is completely covered than raising that entire edge can cause problems for the other edges if the graph is very dense (see Figure 2.1 for an example of this). Therefore the head is only raised if its length is smaller than that of the tail. This

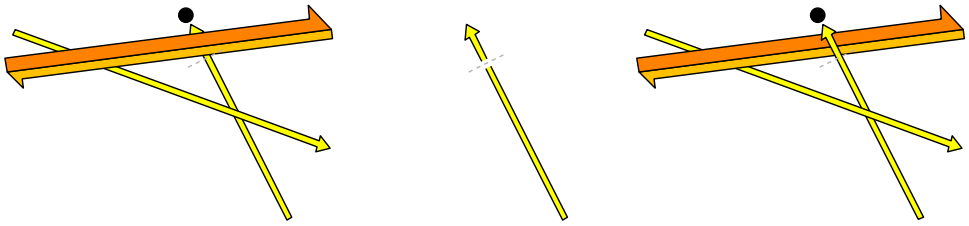


Figure 4.7: Splitting the edge and raising the head of the edge.

method will also work if the edge is bidirectional, but now one has to take both heads into account. For our problem this procedure is needed only once after the optimization process is finished. If we were to simulate the optimization process in real-time then the current solution would become too slow for any practical use. In that case one could use *binary space partition trees* commonly abbreviated to BSP-trees. These BSP-trees partition the objects that need to be drawn efficiently allowing a more fluent computation of the drawing order. BSP-trees are described in detail in by Paterson and Yao [20] and de Berg et al. [10].

The problem of cyclic edge ordering where the edges are not nicely stacked up in order at begin or endpoint can lead to situations which render certain edge heads invisible. An example of this phenomenon is depicted in the left image of Figure 4.8. This can be solved using similar strategy as in the previous case. We calculate the angles of all edges with respect to a given reference, say the x-axis. Go through all edges (counter)clockwise and check whether each consecutive edge of the same class is below the next. If this is not the case and if the angle between the edges is fairly small we can raise that edge in similar to the method shown above.

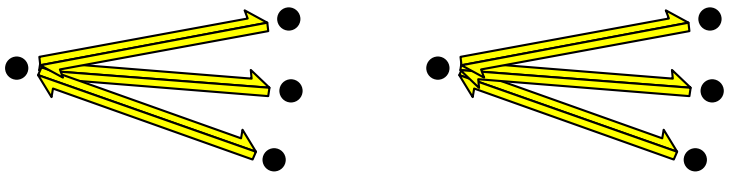


Figure 4.8: On the left a situation where edge weaving occurs. Right shows the result after ordering the edges in height clockwise w.r.t. their angles with a reference.

4.6 Algorithm overview

Here we present a simple version of the algorithm as it was presented above. This algorithm is also the base for the implementation as to be used for our test cases. The pseudo-code our flow mapper is listed in Figure 4.9.

Algorithm OPTIMIZE FLOW MAP(V, E, B, CF)
Input. A set of vertices V, a set of edges E, a set of regions B corresponding to each of the vertices and a set of critical features CF
Output. A drawing of a flow map optimized for the three metrics

```
1.    Compute m_i^max for all regions b_i ∈ B
2.    Set m_i = m_i^max
3.    while max_{1≤i≤n}(m_i) ≥ ε
4.       for each v_i ∈ V
5.          Find closest edge e_j to v_i
6.          Move v_i perpendicular to angle of e_j inside b_i for vertex-edge distance
7.          Find smallest angle of two neighboring edges incident to v_i
8.          Move v_i, v_j and v_k inside b_i, b_j and b_k for angular resolution
9.       for each cf_i ∈ CF
10.         Find closest edge e_j to cf_i
11.         Move vertices of edge e_j inside their respective regions for feature-edge distance
12.      for each v_i ∈ V
13.         Apply cooling schedule on m_i
14.   draw map
```

Figure 4.9: Pseudo-code listing for the implementation of our flow mapper.

Chapter 5

Test and results

Now that we have defined how our algorithm operates we tested our algorithm on three maps and with several different data sets to see how it all comes together in practice. The code for the program was implemented in Java. The region data for the test maps, namely the provinces of the Netherlands, 48 continental states of America and the 12 Federal bank districts of the US are created by hand in a simple ASCII format which the program can read during run-time. For some countries additional regions need to be specified which aren't part of the actual regions, but they do need to be displayed to make the map appear coherent and complete. Examples of these regions are the Frisian Islands in the Netherlands. They are required in order to show a complete and coherent picture of the area in question. To distinguish these regions from the normal regions they are colored with a darker shade of grey in the mapper. All test sets run on a 1.7 GHz Pentium IV laptop running under Windows XP. The execution time of most optimizations including the visualization is less than 1 second. The only ones which take longer are those for the polygonal regions as these regions allow greater movement of the vertices. The output is shown on-screen as a raster image, but a save option is included which can store the map as a vector graphic file along with the statistics in a separate file either in LaTeX format or plain text.

The notation is the same for all maps. At the top the resulting map is shown after completing the optimization one complete with all labels and regions and another without to give a clear view of the end result. The region type is denoted by one of the following abbreviations:

- (C) = Circular regions
- (S) = axis parallel Square regions
- (P) = Polygonal regions

For vertex-edge distance (V-E distance) we list the actual distance measured in pixels, plus the end points of the edge $e = \{u, w\}$ that realizes this distance. Critical feature-edge distance (C-E distance) is enlisted in the same format. Angular resolution is listed as the smallest angle in degrees for two edges with a common endpoint in a vertex together with the endpoints v_j and v_k of those two edges. For each map we also listed some statistical data for a better overview on vertex-edge distance and angular resolution in a separate table. For the sake of simplicity all values receive symmetric arithmetic rounding in order to truncate them to one decimal.

In the following three sections we go through three different test cases. First, a map of the Netherlands divided into the twelve provinces is optimized for a data set concerning the local migration of people. The second test set is the continental American map divided into the 48 states. The data set applied here is the movement one 1 USD bank notes in 1976. For the last data set we reused the data from the previous set, but now for the 12 Federal bank reserve districts. For this case we only generated one map which we compare to the output of Tobler's

mapper [29] using the exact same input. We conclude with a discussion of the results and our findings in Section 5.4.

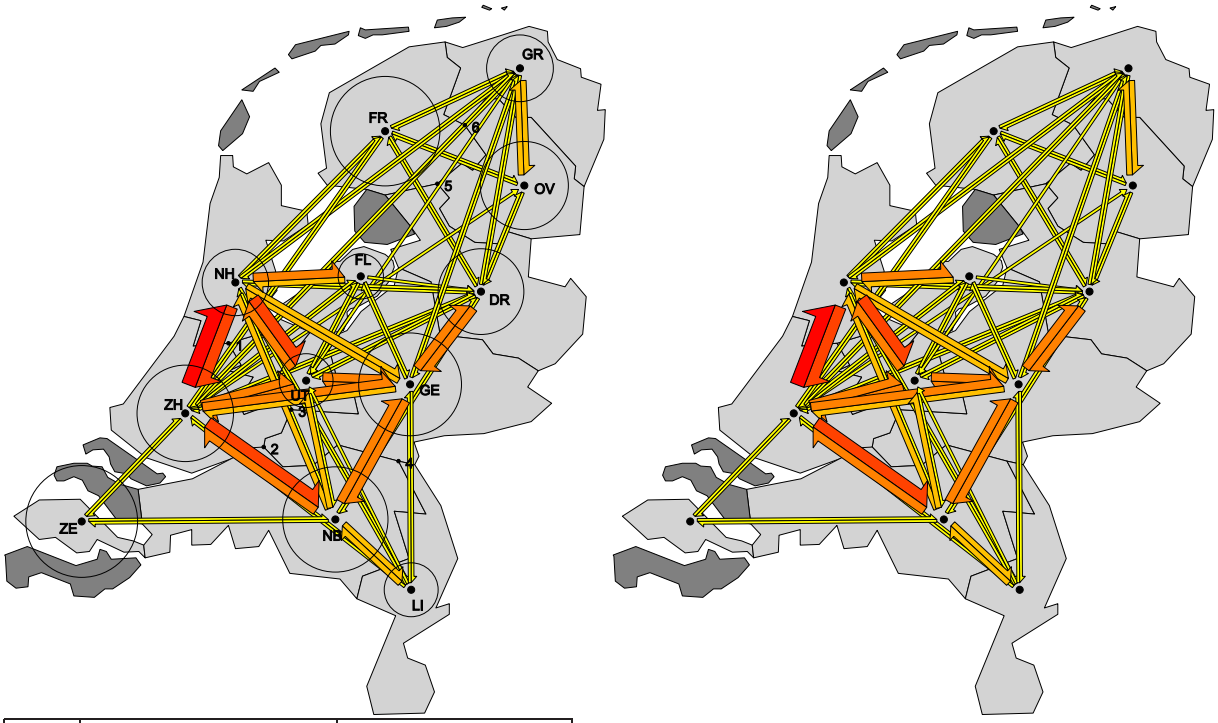
5.1 First test case: Flow map of the Netherlands

We have used two distinct test setups for the analysis of our application. The topic of the first case is about local movement of people between the provinces of the Netherlands in 1996. The data set corresponding to this test case can be obtained from the website of *Centraal Bureau voor de Statistiek* (CBS) [5] and is also shown in Table A.1 in Appendix A.

To obtain the actual data set used for the edge set we applied the average of all values as the threshold, and classified the remaining values on a scale from one to five corresponding to the number of edge classes our implementation can draw. The base map was manually traced from a bitmap and the coordinates were stored in a text file. Notice that the province of Zeeland consists of several disjoint islands introducing another issue for determining the region to which the vertices are confined to. This problem can be solved by determining the region that follows the contours on the set of islands that form Zeeland. This region was traced manually and is used for the computation of the regions to be associated with Zeeland.

5.1.1 Initial map

This is the flow map before applying the optimization algorithm. It shows all the regions and the initial vertex configuration along with all the edges and the critical features marked by the small dots. For these test setups the features appear on top of any edges to show where bad situations may occur.



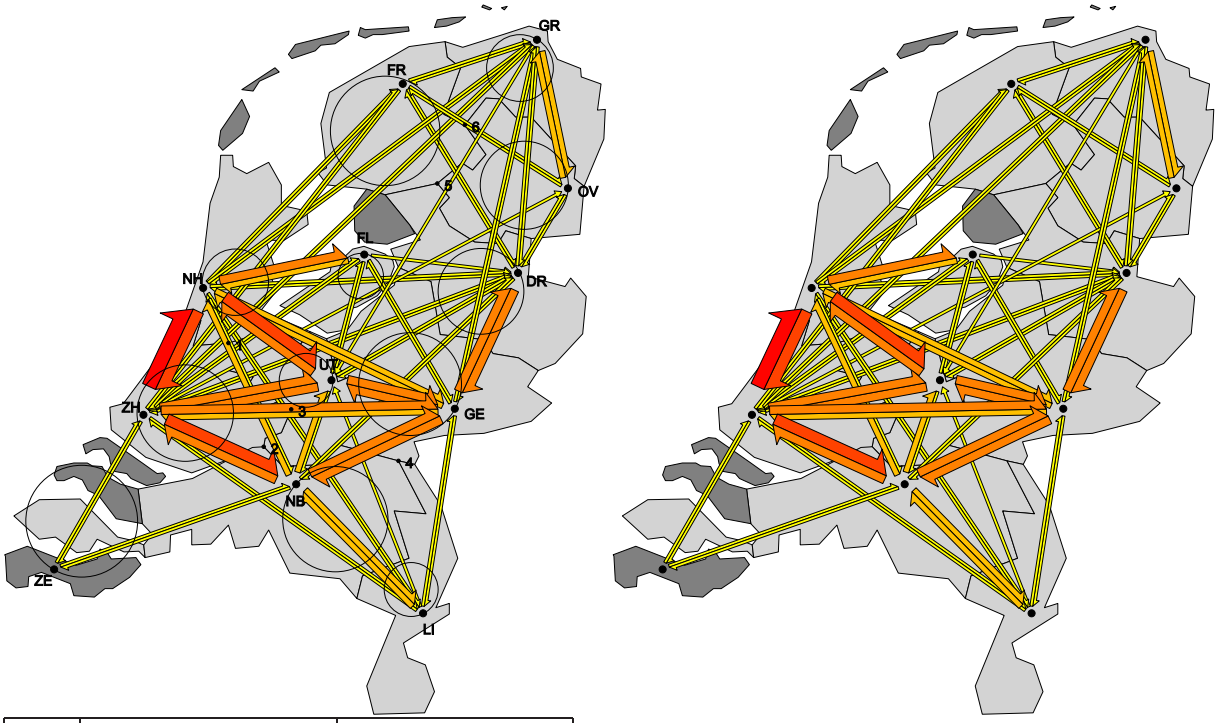
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	148.5	FR	OV	8.9	ZH	NH
FR	39.7	NH	GR	9.4	ZH	NH
OV	30.6	GR	DR	33.9	DR	ZH
DR	47.7	GR	GE	4.6	UT	ZH
FL	11.3	NH	DR	24.4	UT	ZH
GE	9.9	NB	DR	9.4	ZH	UT
UT	12.1	ZH	GE	6.7	FL	GR
NH	45.3	ZH	FR	4.9	FL	DR
ZH	129.5	NB	NH	2.7	NB	LI
ZE	212.1	GR	ZH	45.8	NB	ZH
NB	8.4	ZH	LI	3.6	GE	DR
LI	146.5	DR	NB	3.1	NH	UT

	V-E dist.	Angle
Smallest	8.4	2.7
Largest	212.1	45.8
Average	70.1	13.1
Median	42.5	7.8

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	3.5	ZH	FR	0
3	0.0	NH	NB	1
5	4.2	UT	GR	0
6	0.0	GR	ZH	1

Figure 5.1: Graph of local movements in the Netherlands in 1996 [5] prior to optimization with circular regions along with the metrics and statistics for the map.

5.1.2 VE distance (C)



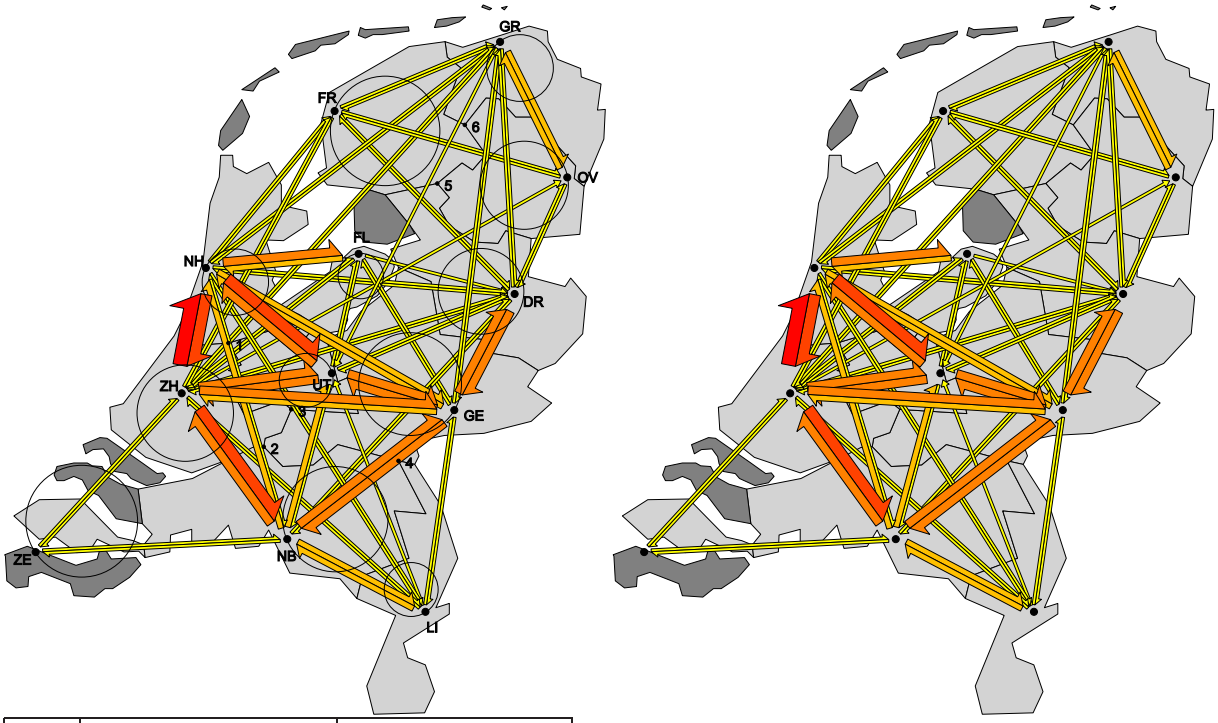
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	150.5	FR	OV	7.0	ZH	NH
FR	59.3	NH	GR	6.3	ZH	NH
OV	56.7	GR	DR	31.4	DR	ZH
DR	41.6	GR	GE	9.2	UT	ZH
FL	32.4	NH	DR	24.2	ZH	NH
GE	73.5	DR	NB	12.4	GR	DR
UT	31.1	GE	NH	16.5	FL	GR
NH	40.3	ZH	FR	8.7	LI	NB
ZH	145.4	NB	NH	7.3	OV	DR
ZE	247.9	LI	ZH	40.6	NB	ZH
NB	41.1	ZH	LI	18.1	DR	GE
LI	230.4	GE	NB	10.1	ZH	NB

	$d_e(v_*)$	ar_*
Smallest	31.1	6.3
Largest	247.9	40.6
Average	95.9	16.0
Median	58.0	11.3

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	NB	NH	1
3	0.0	ZH	GE	1
6	0.0	OV	FR	1

Figure 5.2: Figure 5.1 optimized for VE-distance (C).

5.1.3 VE distance and angular resolution (C)



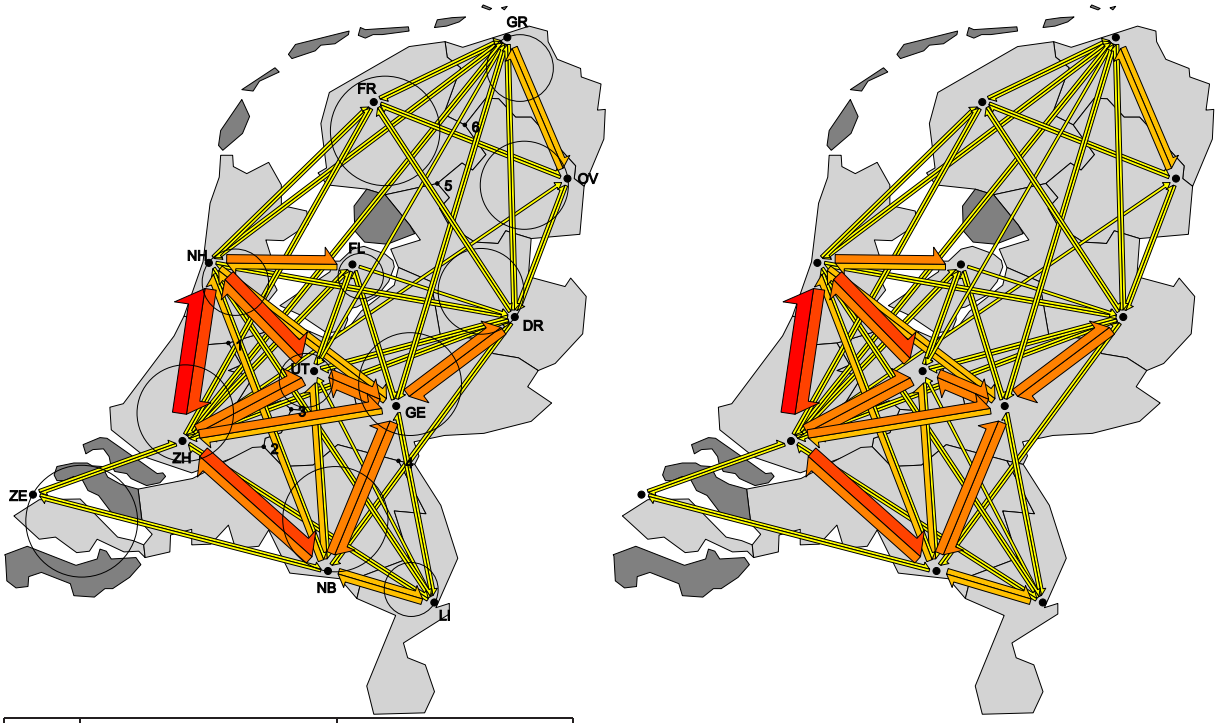
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	154.5	FR	OV	10.3	ZH	NH
FR	61.2	NH	GR	10.9	ZH	NH
OV	80.0	GR	DR	36.5	DR	ZH
DR	60.5	GR	GE	6.8	UT	ZH
FL	34.1	NH	DR	33.0	ZH	NH
GE	45.2	DR	NB	13.0	UT	NH
UT	29.5	ZH	GE	14.3	FL	GR
NH	50.5	ZH	FR	10.1	FL	DR
ZH	75.9	NB	NH	9.0	DR	UT
ZE	287.5	NB	ZH	44.4	NB	ZH
NB	50.0	LI	ZH	9.5	DR	GE
LI	189.2	GE	NB	11.1	NH	UT

	$d_e(v_*)$	ar_*
Smallest	29.5	6.8
Largest	287.5	44.4
Average	93.2	17.4
Median	60.9	11.0

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	NH	NB	2
2	1.7	NH	NB	0
3	1.9	LI	NH	0
4	0.0	GE	NB	1
6	6.7	GR	UT	0

Figure 5.3: Figure 5.1 optimized for VE-distance and angular resolution (C).

5.1.4 VE distance, angular resolution and critical features (C)



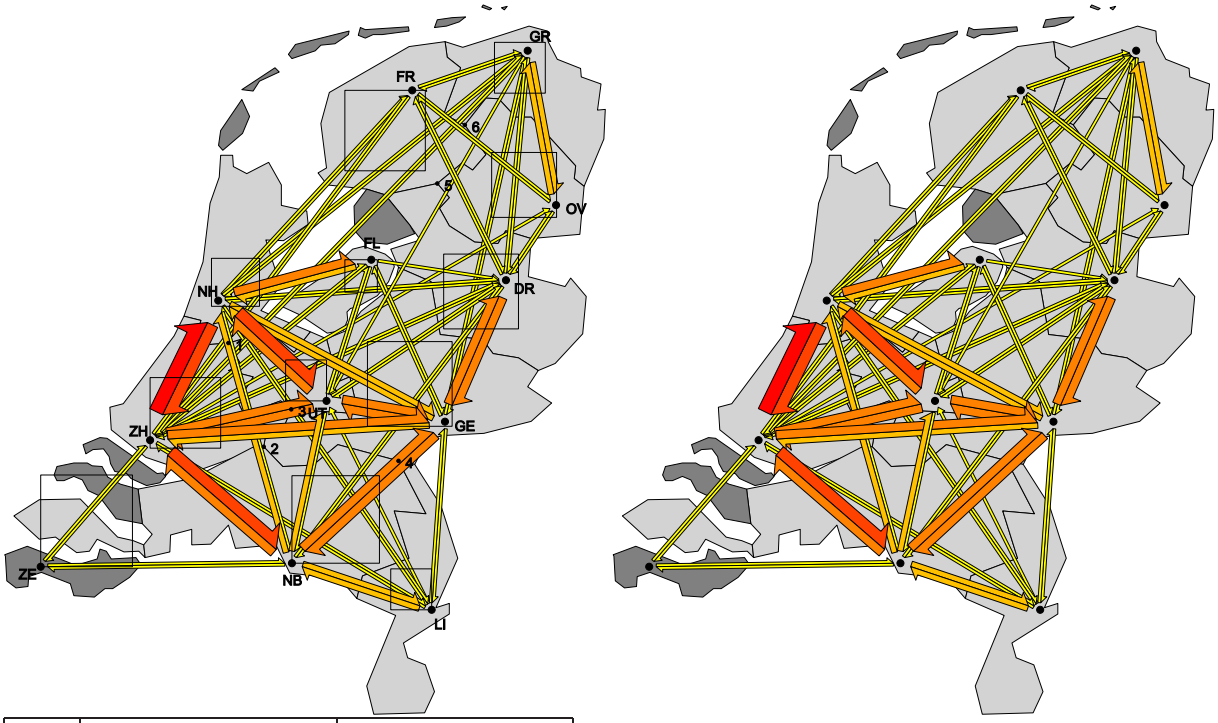
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	150.7	FR	OV	8.8	UT	ZH
FR	37.0	NH	GR	16.3	ZH	NH
OV	75.9	GR	DR	34.9	DR	ZH
DR	120.0	OV	ZH	5.4	ZH	UT
FL	26.7	GR	ZH	24.2	UT	ZH
GE	55.5	DR	ZH	14.4	UT	NH
UT	23.3	OV	ZH	10.3	FL	GR
NH	87.5	ZH	FR	7.2	FR	GR
ZH	117.9	NB	NH	5.1	GR	FL
ZE	225.1	ZH	GR	34.2	NB	ZH
NB	39.8	LI	ZH	13.9	GE	DR
LI	143.8	DR	NB	6.1	NH	UT

	$d_e(v_*)$	ar_*
Smallest	23.3	5.1
Largest	225.1	34.9
Average	91.9	15.1
Median	81.7	12.1

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	7.1	NB	NH	0
3	7.4	ZH	GE	0
4	7.6	LI	GE	0
5	8.0	FR	DR	0
6	8.0	GR	UT	0

Figure 5.4: Figure 5.1 optimized for VE-distance, angular resolution and critical features (C).

5.1.5 VE distance (S)



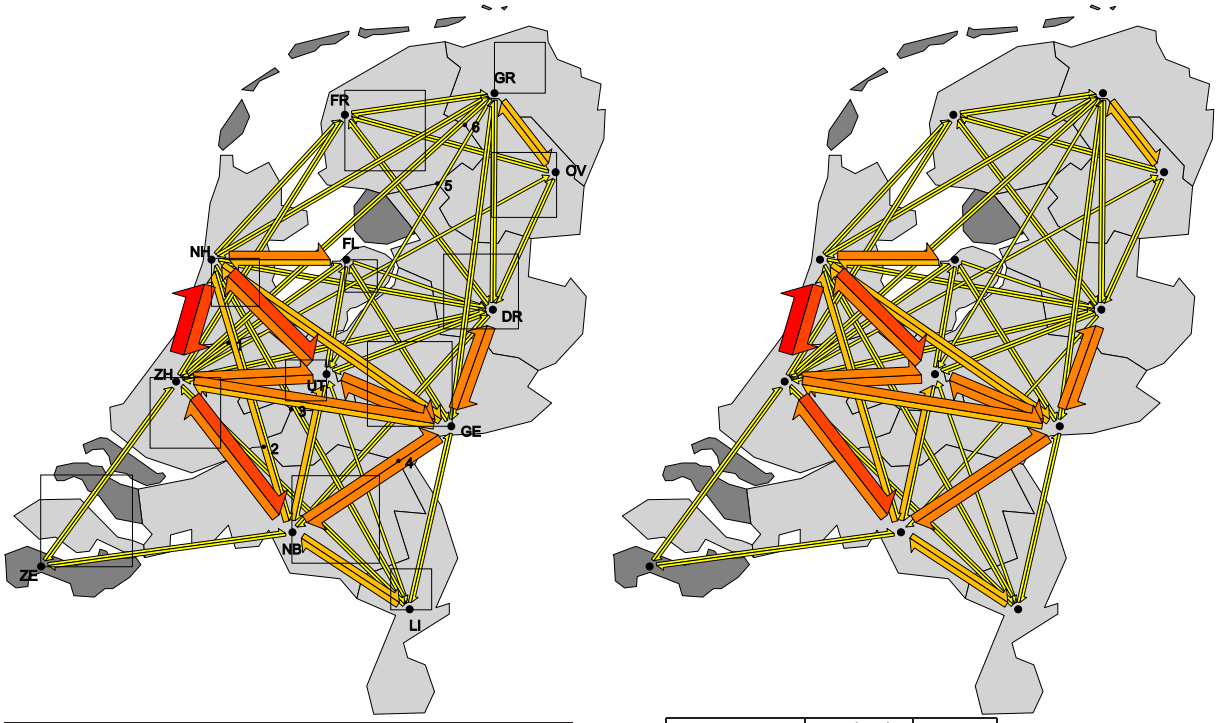
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	142.0	FR	OV	7.0	ZH	NH
FR	55.1	NH	GR	5.9	ZH	NH
OV	56.6	GR	DR	26.2	DR	ZH
DR	36.7	GR	GE	9.7	UT	ZH
FL	38.4	NH	DR	24.3	ZH	NH
GE	48.0	DR	NB	10.8	GR	DR
UT	29.9	ZH	GE	12.3	FL	GR
NH	37.3	ZH	FR	8.4	FR	GR
ZH	138.4	NB	NH	5.9	OV	DR
ZE	225.0	NB	ZH	48.4	NB	ZH
NB	41.6	LI	ZH	10.1	DR	GE
LI	170.9	GE	NB	7.9	NH	UT

	$d_e(v_*)$	ar_*
Smallest	29.9	5.9
Largest	225.0	48.4
Average	84.8	14.7
Median	51.5	9.9

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	NB	NH	1
2	2.1	NH	NB	0
3	0.0	UT	ZH	2
4	0.0	NB	GE	1
6	4.2	FR	OV	0

Figure 5.5: Figure 5.1 optimized for VE-distance (S).

5.1.6 VE distance and angular resolution (S)



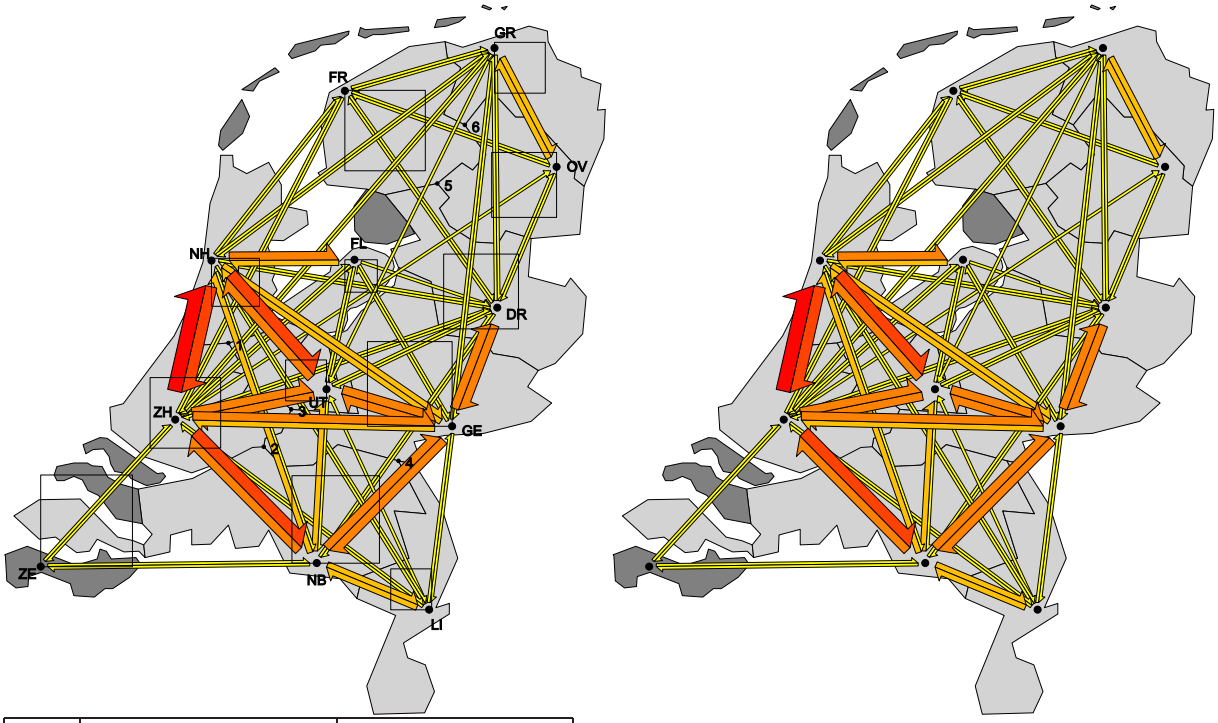
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	81.3	FR	OV	7.0	DR	GE
FR	77.1	NH	GR	10.3	ZH	NH
OV	83.5	GR	DR	36.6	DR	ZH
DR	33.1	GR	GE	8.4	UT	ZH
FL	27.3	NH	DR	35.6	ZH	NH
GE	62.8	DR	NB	12.1	UT	NH
UT	32.2	GE	NH	21.0	FL	GR
NH	46.0	ZH	FR	10.0	FL	DR
ZH	89.2	NB	NH	6.6	GR	FL
ZE	299.6	NB	ZH	46.1	NB	ZH
NB	33.8	LI	ZH	14.3	DR	GE
LI	170.9	GE	NB	10.1	NH	UT

	$d_e(v_*)$	ar_*
Smallest	27.3	6.6
Largest	299.6	46.1
Average	86.6	18.2
Median	69.9	11.2

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	ZH	FL	1
2	0.0	NB	NH	1
3	2.3	LI	NH	0
4	0.0	NB	GE	1
5	3.8	UT	GR	0
6	1.2	GR	ZH	0

Figure 5.6: Figure 5.1 optimized for VE-distance and angular resolution (S).

5.1.7 VE distance, angular resolution and critical features (S)



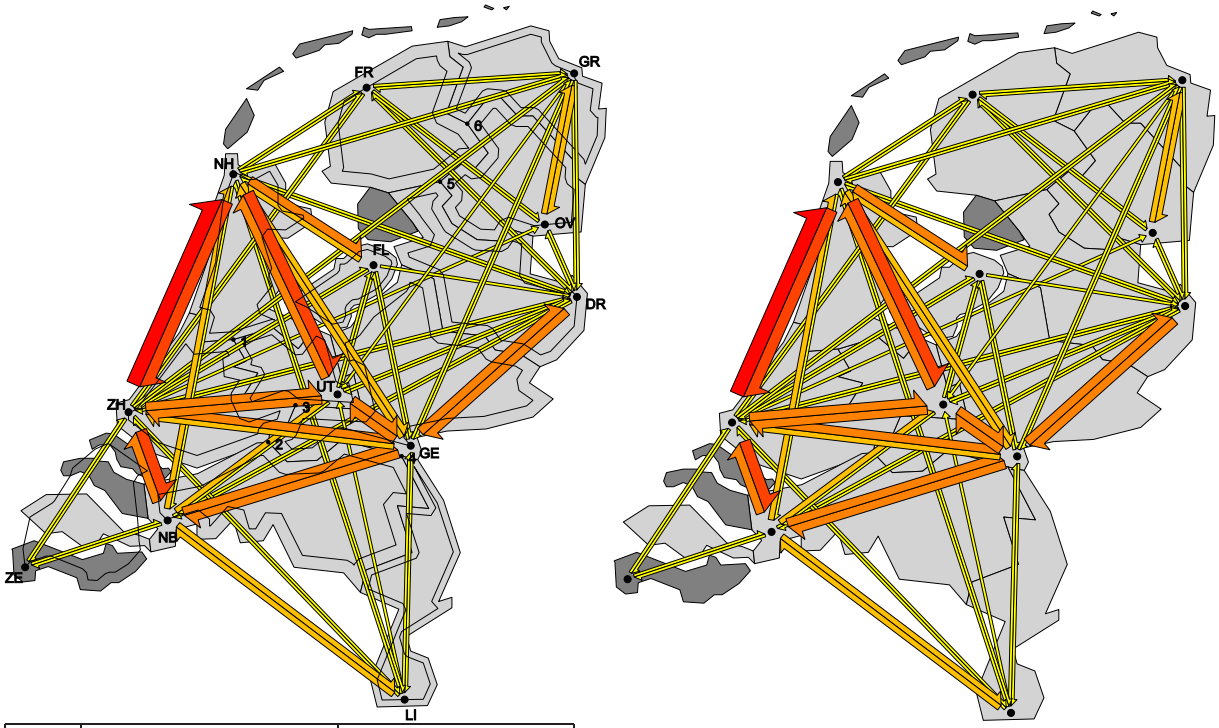
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	124.7	FR	OV	7.0	DR	GE
FR	74.8	NH	GR	10.9	ZH	NH
OV	82.2	GR	DR	33.6	DR	ZH
DR	40.9	GR	GE	6.4	UT	ZH
FL	30.1	NH	DR	36.1	UT	ZH
GE	40.8	DR	NB	14.4	GR	DR
UT	25.8	DR	ZH	14.0	FL	GR
NH	53.7	ZH	FR	9.7	FL	DR
ZH	114.6	NB	NH	7.6	GR	FL
ZE	270.0	NB	ZH	46.8	NB	ZH
NB	38.5	LI	ZH	9.5	DR	GE
LI	147.6	GE	NB	7.0	NH	UT

	$d_e(v_*)$	ar_*
Smallest	25.8	6.4
Largest	270.0	46.8
Average	87.0	16.9
Median	64.2	10.3

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	7.7	NB	NH	0
3	6.1	UT	ZH	0
4	7.0	DR	NB	0
6	8.0	FR	OV	0

Figure 5.7: Figure 5.1 optimized for VE-distance, angular resolution and critical features (S).

5.1.8 VE distance (P)



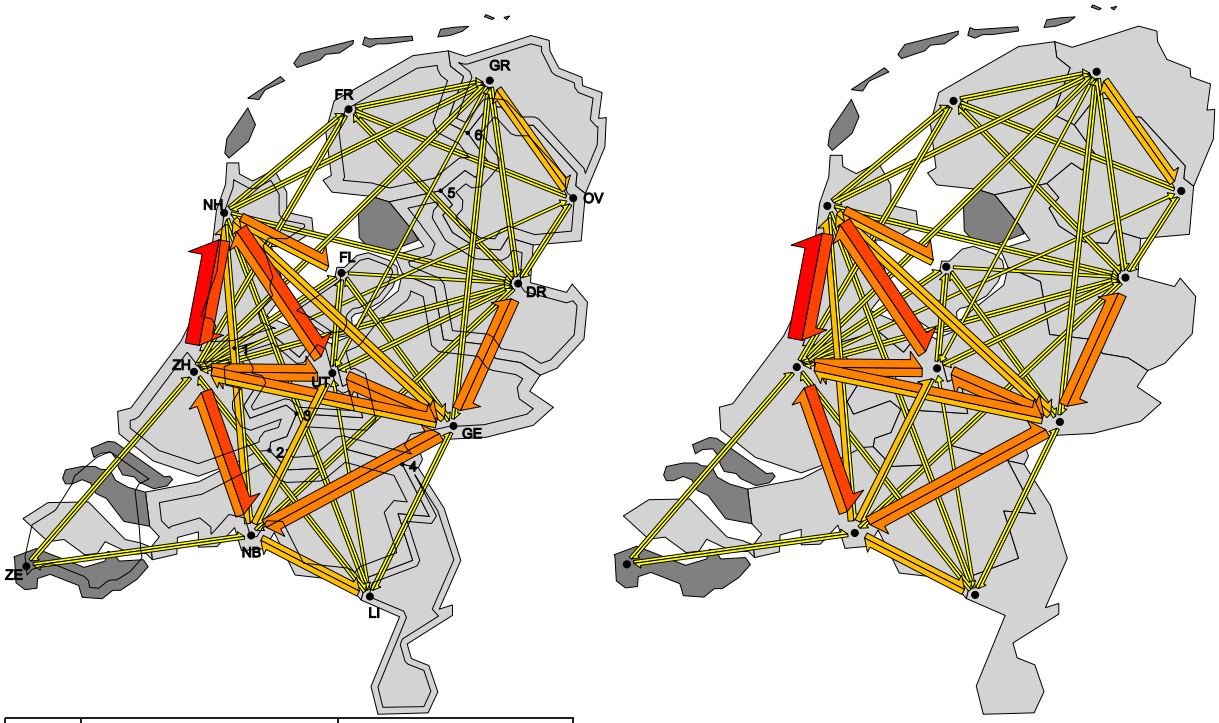
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	193.1	FR	OV	11.7	DR	OV
FR	60.7	NH	GR	7.3	OV	DR
OV	42.1	FR	DR	61.8	ZH	FR
DR	113.4	OV	ZH	6.5	NB	UT
FL	41.9	GR	ZH	27.2	GE	UT
GE	68.9	DR	NB	21.5	NH	FL
UT	41.8	NB	DR	20.8	FL	GR
NH	76.5	ZH	FR	7.3	UT	LI
ZH	75.9	NB	NH	6.3	GR	FL
ZE	202.6	NB	ZH	38.1	NB	ZH
NB	63.0	LI	ZH	7.9	UT	DR
LI	332.8	GE	NB	5.7	NH	UT

	$d_e(v_*)$	ar_*
Smallest	41.8	5.7
Largest	332.8	61.8
Average	108.5	18.5
Median	72.4	9.8

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	4.0	GR	ZH	0
2	0.0	NB	UT	1
3	0.0	UT	ZH	1
4	0.0	GE	NB	1
5	3.2	GR	ZH	0

Figure 5.8: Figure 5.1 optimized for VE-distance (P).

5.1.9 VE distance and angular resolution (P)



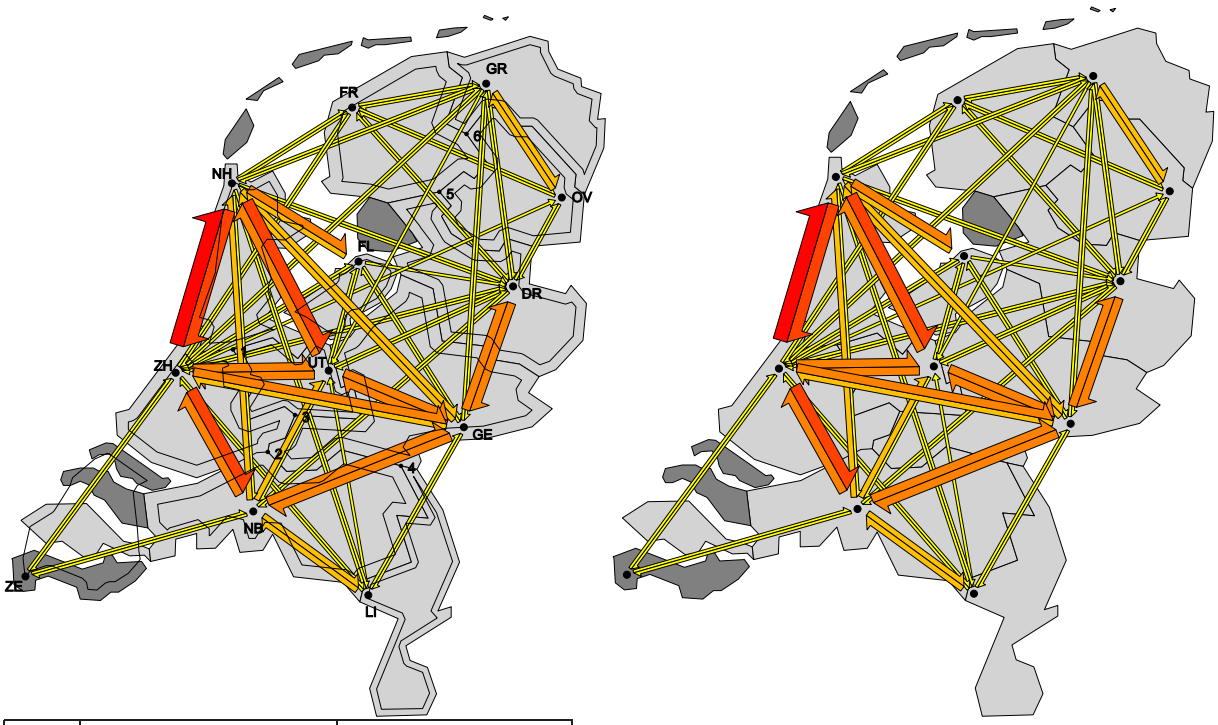
	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	109.1	FR	OV	14.0	DR	GE
FR	49.4	NH	GR	19.7	ZH	NH
OV	91.3	GR	DR	32.6	DR	ZH
DR	67.1	GR	GE	10.1	FL	NH
FL	39.0	ZH	OV	41.4	GE	UT
GE	81.0	DR	NB	10.9	NH	FL
UT	26.8	ZH	GE	23.1	FL	GR
NH	74.5	ZH	FR	13.1	GE	UT
ZH	54.3	NB	NH	9.2	FL	OV
ZE	307.5	NB	ZH	41.4	NB	ZH
NB	76.2	LI	ZH	14.5	ZH	NH
LI	146.3	GE	NB	11.3	NH	UT

	$d_e(v_*)$	ar_*
Smallest	26.8	9.2
Largest	307.5	41.4
Average	93.5	20.1
Median	75.3	14.2

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	NB	NH	1
3	1.2	LI	NH	0
4	1.5	GE	NB	0
6	5.9	GR	UT	0

Figure 5.9: Figure 5.1 optimized for VE-distance and angular resolution (P).

5.1.10 VE distance, angular resolution and critical features (P)



	V-E distance			Smallest Angle		
v_i	$d_e(v_i)$	v	w	ar_i	v_j	v_k
GR	103.3	FR	OV	11.3	DR	GE
FR	34.3	NH	GR	24.1	ZH	NH
OV	82.0	GR	DR	33.2	FR	GR
DR	53.3	GR	GE	10.2	UT	ZH
FL	34.8	ZH	OV	43.4	UT	ZH
GE	102.5	DR	NB	11.1	NH	FL
UT	31.8	ZH	GE	13.4	FL	GR
NH	79.3	ZH	FR	9.0	UT	LI
ZH	90.1	NB	NH	6.9	FL	OV
ZE	318.6	NB	ZH	37.7	NB	ZH
NB	42.1	LI	ZH	19.1	DR	GE
LI	160.7	GE	NB	8.3	NH	UT

	$d_e(v_*)$	ar_*
Smallest	31.8	6.9
Largest	318.6	43.4
Average	94.4	19.0
Median	80.7	12.4

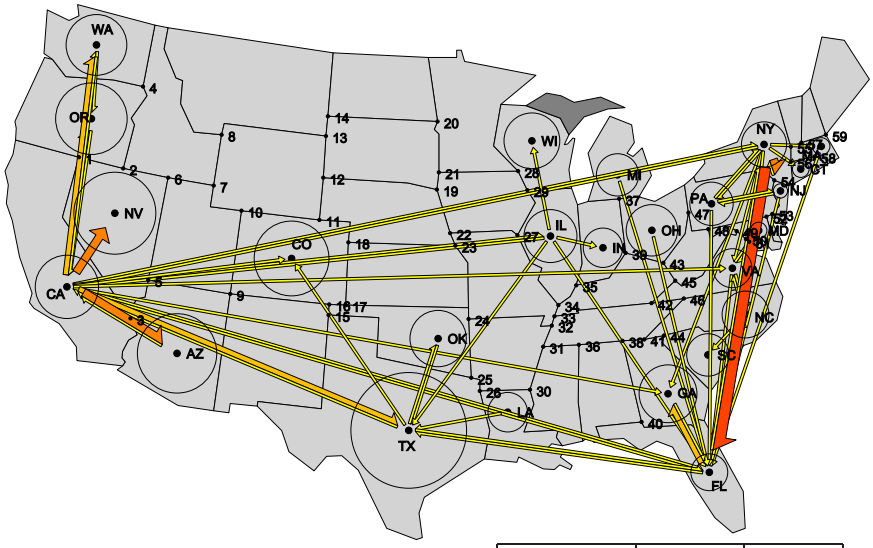
	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	3.5	OV	ZH	0
3	4.6	NB	UT	0
4	6.0	GE	NB	0
6	8.0	GR	UT	0

Figure 5.10: Figure 5.1 optimized for VE-distance, angular resolution and critical features (P).

5.2 Second test case: Flow map of the USA

The second case analysis is about the movement of 1-US-Dollar banknotes between the individual counties of the USA in 1976. The data sets used for the American maps originate from the Excel tables that come along W. Tobler's implementation of the mapper [29]. Using the same method as for the Dutch data input the set of edges is obtained for the five distinct width classes. For this map we only work with the 48 states on the mainland of America also referred to as the continental United States. Because a table for the properties of all regions would become too large to display nicely, we've opted to show the vertex-edge distance and angular resolution for the vertices which have any edges attached to them, and we only show the complete map with the interior regions and all of its labels. All state names are abbreviated to match the two character United States Postal Service (USPS) codes [26]. The input table after applying the median as threshold can be found in Appendix B in Table B.1.

5.2.1 Initial map



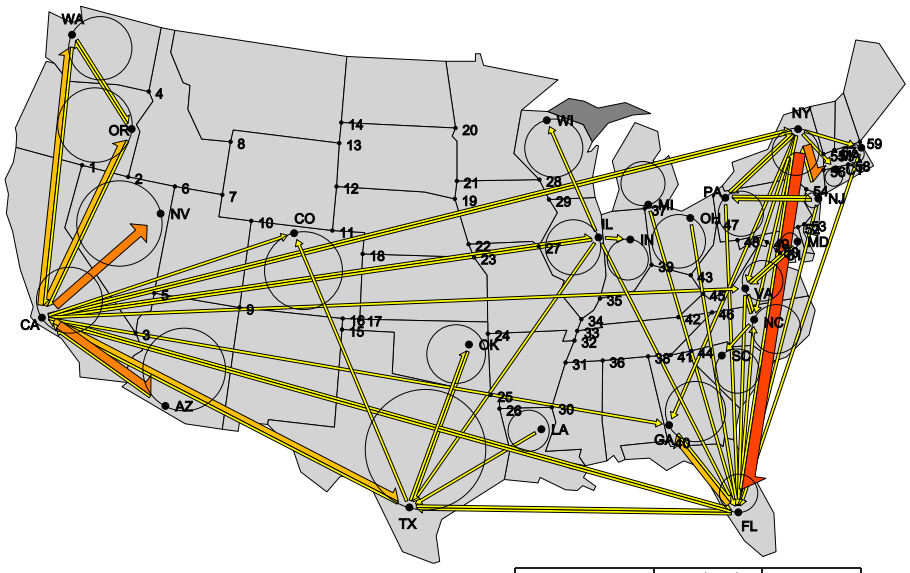
	$d_e(v_*)$	ar_*
Smallest	0.0	1.2
Largest	257.3	175.6
Average	42.6	27.3
Median	22.9	30.0

	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	24.3	TX	CA	N/A	-	-
CA	257.3	OR	WA	1.2	CO	IL
CO	3.2	CA	IL	117.1	CA	TX
CT	18.2	FL	MA	N/A	-	-
FL	134.3	GA	CA	1.3	NC	NJ
GA	12.4	IL	FL	101.0	CA	NY
IL	54.3	CA	VA	43.5	IN	FL
IN	34.4	CA	VA	N/A	-	-
LA	0.0	CA	FL	N/A	-	-
MD	0.0	NY	FL	N/A	-	-
MA	46.4	NY	CT	73.0	FL	NY
MI	0.0	NY	CA	N/A	-	-
NV	51.8	OR	CA	N/A	-	-
NJ	20.1	NY	FL	65.2	FL	PA
NY	69.8	PA	NJ	3.3	NC	FL
NC	0.0	NY	FL	22.1	VA	NY
OH	22.9	MI	FL	N/A	-	-
OK	19.3	CA	GA	N/A	-	-
OR	1.9	WA	CA	175.6	CA	WA
PA	41.9	GA	NY	37.9	NY	NJ
SC	3.6	FL	PA	N/A	-	-
TX	61.3	FL	CA	18.3	LA	FL
VA	12.4	FL	NY	20.5	NY	MD
WA	112.0	CA	OR	3.0	CA	OR
WI	71.7	CA	NY	N/A	-	-

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	0.0	CA	WA	1
3	0.0	CA	AZ	1
5	0.0	IL	CA	1
15	0.0	TX	CO	1
23	0.0	CA	IL	1
27	2.7	CA	IL	0
29	0.0	CA	NY	1
38	0.0	IL	FL	1
43	3.5	OH	FL	0
44	4.5	FL	MI	0
48	4.6	FL	PA	0
49	6.5	NY	GA	0
50	0.0	FL	NY	1
51	0.0	NY	FL	2
52	5.5	NY	FL	0
53	3.3	FL	NJ	0
54	0.0	NJ	NY	1
55	2.0	MA	NY	0
56	0.0	NY	CT	1
57	2.6	NY	MA	0
58	4.6	FL	MA	0

Figure 5.11: Movement of 1 USD bank notes between each of the 48 states in the United States of America prior to optimization with circular regions along with the metrics and statistics for the map.

5.2.2 VE distance (C)



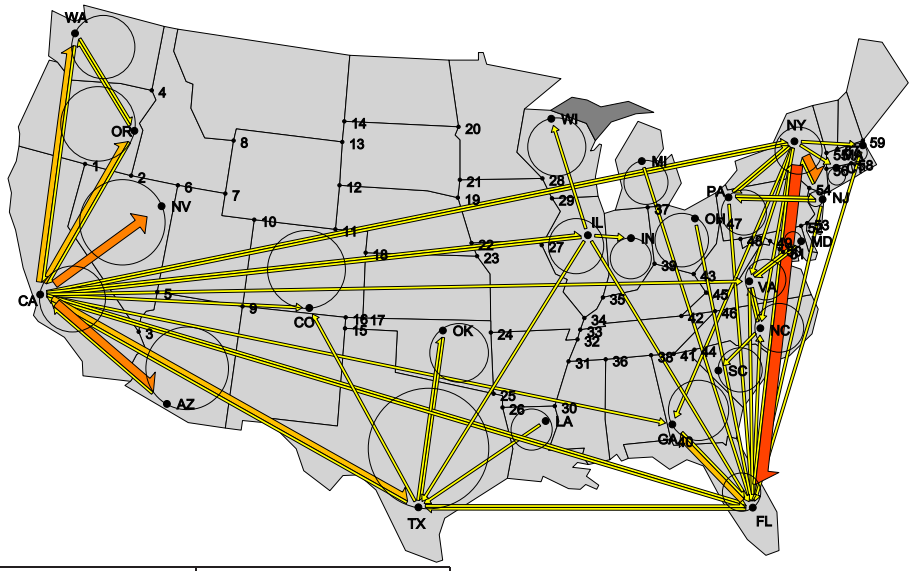
	$d_e(v_*)$	ar_*
Smallest	8.5	2.9
Largest	303.7	122.2
Average	69.4	27.3
Median	49.8	41.1

	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	27.7	TX	CA	N/A	-	-
CA	303.7	OR	WA	4.5	CO	NY
CO	26.4	CA	NY	94.3	CA	TX
CT	24.0	NY	NJ	N/A	-	-
FL	161.9	GA	CA	2.9	VA	NC
GA	32.3	FL	IL	103.8	CA	NY
IL	78.6	NY	CA	46.8	TX	CA
IN	37.8	IL	FL	N/A	-	-
LA	35.2	CA	FL	N/A	-	-
MD	8.5	NY	FL	N/A	-	-
MA	48.9	NY	CT	87.7	FL	NY
MI	49.8	NY	CA	N/A	-	-
NV	86.6	CA	CO	N/A	-	-
NJ	28.7	NY	FL	73.2	PA	NY
NY	95.7	FL	MA	4.1	FL	NC
NC	12.3	VA	FL	28.9	VA	NY
OH	51.2	MI	FL	N/A	-	-
OK	64.3	CA	GA	N/A	-	-
OR	96.8	WA	CA	122.2	CA	WA
PA	57.0	GA	NY	43.9	NY	NJ
SC	11.0	OH	FL	N/A	-	-
TX	117.9	FL	CA	14.9	OK	IL
VA	20.2	NY	GA	17.7	NC	FL
WA	162.1	CA	OR	38.4	CA	OR
WI	96.8	CA	NY	N/A	-	-

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
3	0.0	CA	GA	1
5	0.9	NY	CA	0
9	0.8	CA	VA	0
23	0.0	IL	CA	1
25	0.5	GA	CA	0
27	0.0	IL	CA	1
29	7.8	NY	CA	0
30	3.2	GA	CA	0
35	5.9	VA	CA	0
37	6.6	FL	MI	0
40	7.5	FL	GA	0
42	4.0	FL	MI	0
44	0.0	MI	FL	1
45	0.0	OH	FL	1
46	7.1	OH	FL	0
49	6.9	NY	VA	0
50	0.0	FL	NY	1
51	0.0	NY	FL	1
52	5.8	NY	FL	0
55	0.2	NY	CT	0
56	6.8	NY	NJ	0
59	5.7	NY	MA	0

Figure 5.12: Figure 5.11 optimized for VE-distance (C).

5.2.3 VE distance and angular resolution (C)



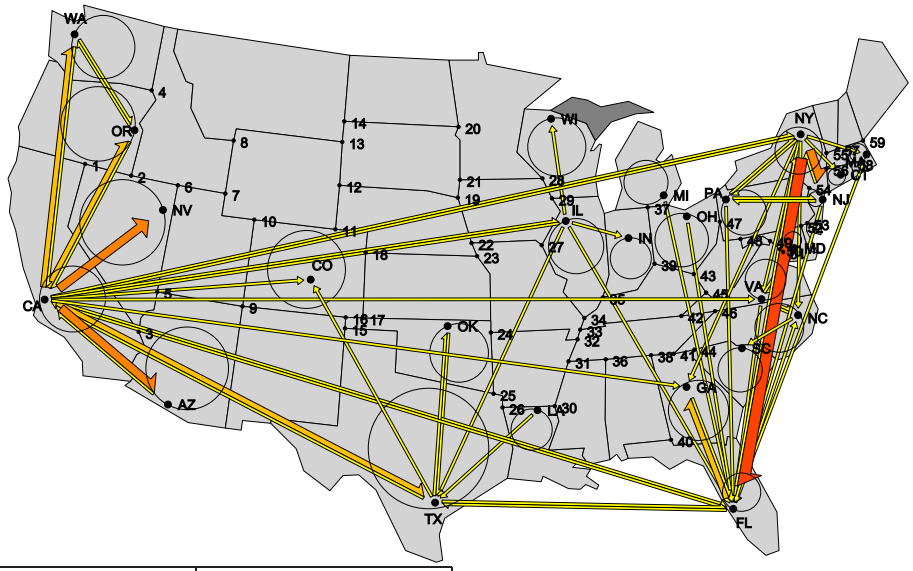
	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	44.2	TX	CA	N/A	-	-
CA	274.6	OR	WA	3.9	VA	CO
CO	26.8	VA	CA	121.5	TX	CA
CT	26.1	NY	NJ	N/A	-	-
FL	168.2	GA	CA	3.3	VA	NC
GA	37.4	FL	IL	101.7	CA	NY
IL	69.4	CA	VA	51.9	TX	CA
IN	49.1	FL	MI	N/A	-	-
LA	30.4	CA	FL	N/A	-	-
MD	10.2	NY	FL	N/A	-	-
MA	50.4	NY	CT	76.4	FL	NY
MI	12.0	CA	NY	N/A	-	-
NV	84.9	OR	CA	N/A	-	-
NJ	33.0	FL	MA	62.9	PA	NY
NY	79.5	PA	NJ	3.8	FL	NC
NC	13.0	VA	FL	23.6	VA	NY
OH	46.2	MI	FL	N/A	-	-
OK	63.2	VA	CA	N/A	-	-
OR	100.3	WA	CA	118.7	CA	WA
PA	55.7	GA	NY	41.6	NY	NJ
SC	9.9	FL	OH	N/A	-	-
TX	134.8	FL	CA	23.9	IL	LA
VA	18.3	NY	GA	12.4	NC	FL
WA	166.0	CA	OR	38.9	CA	OR
WI	98.5	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	9.9	3.3
Largest	274.6	121.5
Average	68.1	27.4
Median	49.1	40.3

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
3	6.8	FL	CA	0
5	0.0	CA	VA	1
9	2.5	CO	CA	0
11	3.5	CA	NY	0
18	5.5	CA	IL	0
22	2.9	CA	IL	0
25	3.4	IL	TX	0
27	3.1	IL	CA	0
29	7.1	NY	CA	0
45	5.2	FL	OH	0
46	0.2	OH	FL	0
50	0.0	FL	NY	1
51	0.0	NY	FL	2
52	7.4	NY	FL	0
54	6.2	NJ	NY	0
57	7.3	MA	NY	0
59	4.9	NY	MA	0

Figure 5.13: Figure 5.11 optimized for VE-distance and angular resolution (C).

5.2.4 VE distance, angular resolution and critical features (C)



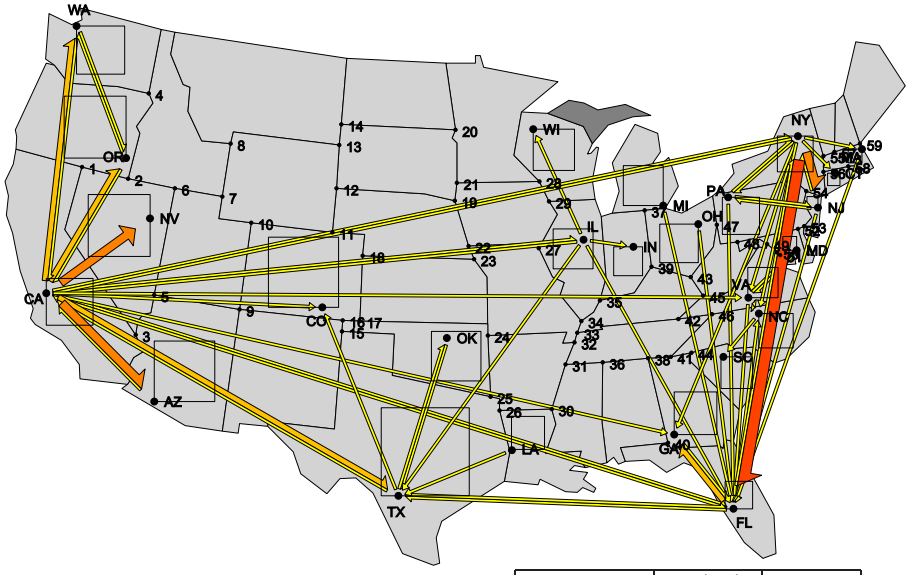
	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	48.6	TX	CA	N/A	-	-
CA	278.7	OR	WA	3.8	NY	IL
CO	24.9	IL	CA	114.9	CA	TX
CT	24.4	NY	NJ	N/A	-	-
FL	189.8	GA	CA	2.1	NC	MA
GA	28.0	FL	MI	106.6	CA	NY
IL	46.2	NY	CA	44.6	IN	FL
IN	63.5	FL	MI	N/A	-	-
LA	50.5	CA	FL	N/A	-	-
MD	4.3	NY	FL	N/A	-	-
MA	47.7	NY	CT	86.4	FL	NY
MI	40.1	NY	CA	N/A	-	-
NV	86.6	CA	NY	N/A	-	-
NJ	31.4	NY	NC	71.3	PA	NY
NY	90.9	PA	NJ	3.1	FL	VA
NC	10.3	NJ	FL	41.0	FL	SC
OH	24.1	MI	FL	N/A	-	-
OK	38.8	CA	GA	N/A	-	-
OR	98.2	WA	CA	120.0	CA	WA
PA	60.0	GA	NY	41.1	NY	NJ
SC	16.2	PA	FL	N/A	-	-
TX	113.4	FL	CA	20.8	OK	IL
VA	9.5	FL	NY	19.4	NY	MD
WA	164.4	CA	OR	38.5	CA	OR
WI	95.5	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	4.3	2.1
Largest	278.7	120.0
Average	67.4	28.5
Median	47.7	41.0

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
3	1.0	FL	CA	0
5	1.1	CO	CA	0
11	5.5	CA	NY	0
18	0.0	IL	CA	1
22	7.3	IL	CA	0
35	0.7	CA	VA	0
38	7.1	IL	FL	0
43	3.2	FL	OH	0
44	5.0	FL	MI	0
45	7.6	CA	VA	0
46	7.6	GA	NY	0
49	7.5	VA	NY	0
50	0.0	FL	NY	1
51	0.0	NY	FL	1
52	0.4	NY	NC	0
55	5.5	NY	CT	0
56	7.5	NY	NJ	0
57	4.5	MA	NY	0

Figure 5.14: Figure 5.11 optimized for VE-distance, angular resolution and critical features (C).

5.2.5 VE distance (S)



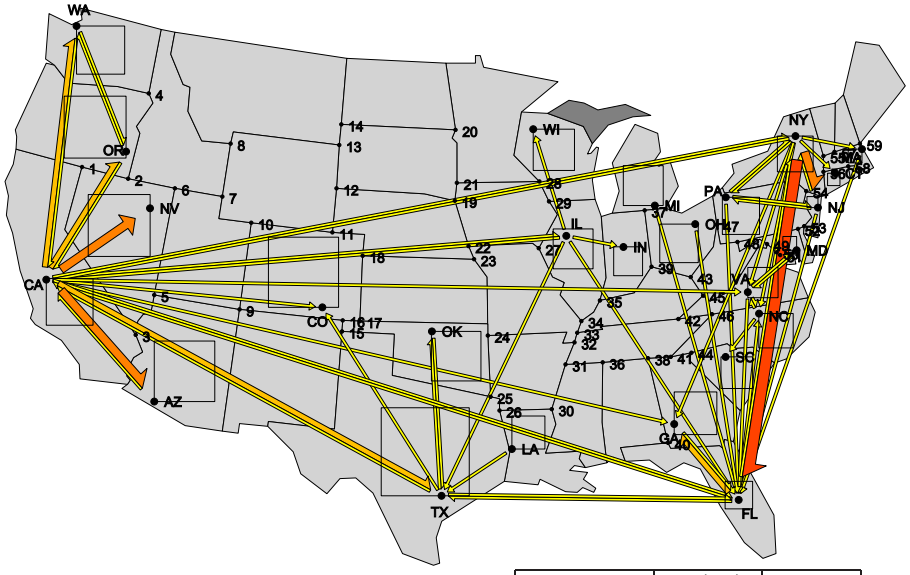
	$d_e(v_*)$	ar_*
Smallest	8.0	2.4
Largest	228.2	128.9
Average	65.8	28.1
Median	54.7	39.8

	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	54.3	TX	CA	N/A	-	-
CA	228.2	OR	WA	2.6	VA	CO
CO	18.0	VA	CA	114.9	TX	CA
CT	25.0	FL	MA	N/A	-	-
FL	138.0	GA	CA	2.4	NC	NY
GA	22.8	FL	IL	99.8	CA	NY
IL	79.7	NY	CA	48.4	TX	CA
IN	55.8	FL	MI	N/A	-	-
LA	9.6	FL	CA	N/A	-	-
MD	10.4	NY	FL	N/A	-	-
MA	52.4	NY	CT	81.8	FL	NY
MI	54.7	NY	CA	N/A	-	-
NV	70.6	OR	CA	N/A	-	-
NJ	30.4	NY	FL	67.8	PA	NY
NY	94.0	FL	MA	2.6	FL	NC
NC	8.0	FL	NY	31.9	FL	SC
OH	41.6	MI	FL	N/A	-	-
OK	61.4	VA	CA	N/A	-	-
OR	89.2	WA	CA	128.9	CA	WA
PA	58.4	OH	FL	47.8	NY	NJ
SC	10.6	OH	FL	N/A	-	-
TX	123.8	FL	CA	18.8	OK	IL
VA	19.8	NC	NY	28.4	NY	MD
WA	205.1	CA	OR	27.0	CA	OR
WI	84.6	CA	NY	N/A	-	-

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
3	4.4	CA	TX	0
5	1.5	VA	CA	0
11	0.0	CA	NY	1
18	4.6	CA	IL	0
19	6.1	CA	NY	0
22	3.3	CA	IL	0
25	5.0	GA	CA	0
27	1.8	IL	CA	0
30	3.0	GA	CA	0
35	5.4	VA	CA	0
38	1.9	FL	IL	0
40	6.5	FL	GA	0
45	0.6	CA	VA	0
46	1.8	OH	FL	0
49	1.1	NY	VA	0
50	0.0	FL	NY	1
51	0.0	NY	FL	1
52	7.1	NY	FL	0
54	5.9	NJ	NY	0
55	3.7	NY	CT	0
59	5.2	NY	MA	0

Figure 5.15: Figure 5.11 optimized for VE-distance (S).

5.2.6 VE distance and angular resolution (S)



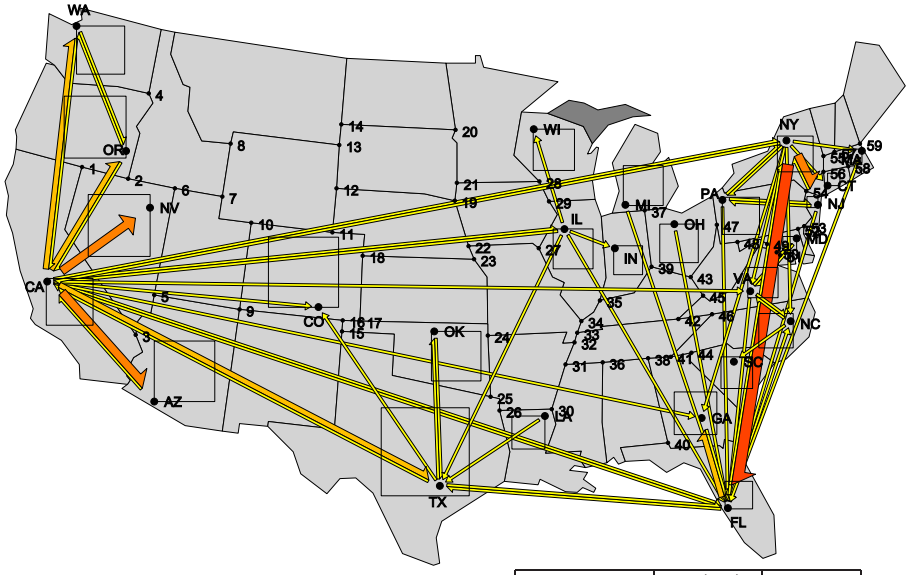
v_i	$d_e(v_i)$ dist.	u	w	ar_i angle	v_j	v_k
AZ	76.2	TX	CA	N/A	-	-
CA	219.4	OR	WA	4.7	GA	FL
CO	33.1	VA	CA	128.0	TX	CA
CT	25.2	FL	MA	N/A	-	-
FL	144.7	GA	CA	2.6	NC	NY
GA	18.2	FL	IL	99.9	CA	NY
IL	75.2	CA	VA	45.8	IN	FL
IN	58.7	IL	FL	N/A	-	-
LA	25.5	FL	CA	N/A	-	-
MD	11.3	NY	FL	N/A	-	-
MA	52.4	NY	CT	81.8	FL	NY
MI	56.7	NY	CA	N/A	-	-
NV	69.2	OR	CA	N/A	-	-
NJ	32.2	FL	MA	66.2	PA	NY
NY	95.7	PA	NJ	2.7	FL	NC
NC	8.5	FL	NY	31.5	FL	SC
OH	46.1	FL	PA	N/A	-	-
OK	50.3	CA	GA	N/A	-	-
OR	89.1	WA	CA	126.6	CA	WA
PA	58.8	GA	NY	47.7	NY	NJ
SC	10.2	FL	PA	N/A	-	-
TX	121.3	FL	CA	29.0	CO	OK
VA	22.2	NY	GA	30.2	NC	FL
WA	196.2	CA	OR	28.3	CA	OR
WI	77.6	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	8.5	2.6
Largest	219.4	128.0
Average	67.0	28.7
Median	56.7	38.6

cf_i	C-E distance cf_d	u	w	Overlap cf_o
3	4.1	TX	CA	0
5	6.5	CO	CA	0
11	6.8	NY	CA	0
15	5.4	CO	TX	0
18	0.0	IL	CA	1
19	0.0	CA	NY	1
22	0.0	IL	CA	1
25	0.0	IL	TX	1
28	1.2	CA	NY	0
29	7.1	IL	WI	0
38	0.3	IL	FL	0
44	7.5	FL	MI	0
45	4.9	FL	OH	0
46	1.7	OH	FL	0
49	4.3	NY	VA	0
50	0.0	FL	NY	1
51	0.0	NY	FL	1
54	5.0	NJ	NY	0
55	4.7	NY	CT	0
59	5.3	NY	MA	0

Figure 5.16: Figure 5.11 optimized for VE-distance and angular resolution (S).

5.2.7 VE distance, angular resolution and critical features (S)



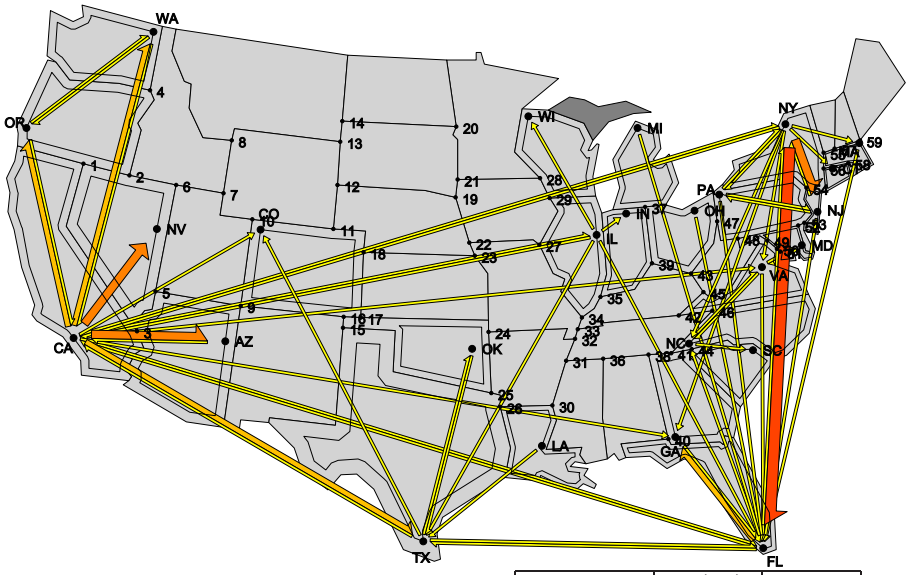
v_i	$d_e(v_i)$ dist.	u	w	ar_i angle	v_j	v_k
AZ	78.9	TX	CA	N/A	-	-
CA	221.8	OR	WA	4.6	VA	CO
CO	31.6	VA	CA	129.5	TX	CA
CT	12.9	NY	NJ	N/A	-	-
FL	136.4	GA	CA	2.0	NC	MA
GA	3.9	MI	FL	95.2	CA	NY
IL	62.0	NY	CA	29.1	IN	FL
IN	38.5	IL	FL	N/A	-	-
LA	39.0	CA	FL	N/A	-	-
MD	9.5	NY	NC	N/A	-	-
MA	68.9	NY	CT	77.2	FL	NY
MI	44.0	NY	CA	N/A	-	-
NV	68.4	OR	CA	N/A	-	-
NJ	31.4	CT	NY	60.9	PA	NY
NY	87.1	PA	NJ	3.5	VA	GA
NC	8.2	NJ	FL	35.9	FL	SC
OH	56.4	MI	FL	N/A	-	-
OK	41.9	CA	GA	N/A	-	-
OR	88.3	WA	CA	127.3	CA	WA
PA	63.4	NY	CA	45.8	NY	NJ
SC	10.5	PA	FL	N/A	-	-
TX	97.6	FL	CA	28.0	OK	IL
VA	11.7	NY	GA	27.8	NY	MD
WA	195.4	CA	OR	28.2	CA	OR
WI	81.0	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	3.9	2.0
Largest	221.8	129.5
Average	63.5	27.8
Median	56.4	32.5

cf_i	C-E distance cf_d	u	w	Overlap cf_o
3	5.3	TX	CA	0
5	4.8	CO	CA	0
11	3.4	NY	CA	0
18	4.9	IL	CA	0
19	0.4	CA	NY	0
22	6.6	IL	CA	0
25	7.8	IL	TX	0
28	4.8	CA	NY	0
29	7.6	IL	WI	0
35	7.9	FL	IL	0
39	5.2	MI	FL	0
43	7.2	OH	FL	0
45	7.4	VA	CA	0
49	0.1	FL	NY	0
50	0.0	NY	FL	1
51	0.0	VA	MD	1
54	2.4	NJ	NY	0
56	7.1	NY	CT	0
57	6.9	MA	NY	0
59	8.0	NY	MA	0

Figure 5.17: Figure 5.11 optimized for VE-distance, angular resolution and critical features (S).

5.2.8 VE distance (P)



v_i	$d_e(v_i)$ dist.	u	w	ar_i angle	v_j	v_k
AZ	27.4	VA	CA	N/A	-	-
CA	312.5	TX	CO	5.3	IL	VA
CO	68.8	CA	NY	87.5	CA	TX
CT	29.3	FL	MA	N/A	-	-
FL	206.4	GA	CA	3.3	VA	NY
GA	37.1	FL	IL	100.1	CA	NY
IL	69.4	CA	VA	49.4	TX	CA
IN	51.6	FL	MI	N/A	-	-
LA	44.3	GA	CA	N/A	-	-
MD	14.9	FL	NJ	N/A	-	-
MA	53.7	NY	CT	91.0	FL	NY
MI	52.7	CA	NY	N/A	-	-
NV	73.3	WA	CA	N/A	-	-
NJ	36.1	FL	MA	59.9	PA	NY
NY	111.1	FL	MA	4.3	GA	NC
NC	18.2	FL	MI	19.9	NY	VA
OH	38.6	FL	PA	N/A	-	-
OK	75.1	VA	CA	N/A	-	-
OR	135.7	CA	WA	114.5	CA	WA
PA	42.9	OH	FL	56.7	NY	NJ
SC	13.6	FL	VA	N/A	-	-
TX	130.8	FL	CA	15.3	OK	IL
VA	19.2	FL	NY	40.5	NC	CA
WA	232.0	OR	CA	38.3	OR	CA
WI	114.7	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	13.6	3.3
Largest	312.5	114.5
Average	80.4	27.4
Median	52.7	44.9

cf_i	C-E distance cf_d	u	w	Overlap cf_o
3	0.0	CA	VA	2
5	1.1	CO	CA	0
9	0.0	IL	CA	1
18	0.0	NY	CA	1
23	0.5	CA	IL	0
26	0.0	IL	TX	1
27	0.0	CA	IL	1
29	0.2	NY	CA	0
40	4.6	GA	CA	0
43	0.0	CA	VA	1
44	1.0	NC	FL	0
46	3.6	FL	OH	0
47	7.8	FL	PA	0
48	1.5	NY	NC	0
49	0.0	NY	VA	1
50	0.0	FL	NY	1
51	0.0	NY	FL	1
54	0.0	NJ	NY	1
55	5.8	NY	CT	0
59	2.1	NY	MA	0

Figure 5.18: Figure 5.11 optimized for VE-distance (P).

5.2.9 VE distance and angular resolution (P)

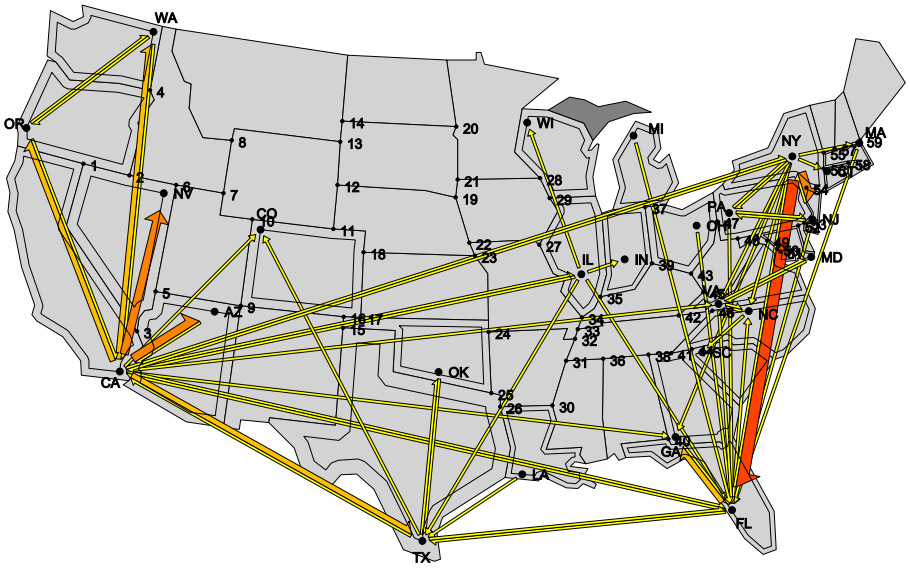


Figure 5.19: Figure 5.11 optimized for VE-distance and angular resolution (P). Statistics on page 61.

	$d_e(v_*)$	ar_*
Smallest	10.8	3.4
Largest	275.2	106.1
Average	78.6	26.8
Median	51.6	46.7

	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	36.6	CO	CA	N/A	-	-
CA	275.2	TX	CO	5.5	IL	VA
CO	130.4	CA	NY	72.2	CA	TX
CT	29.9	NY	NJ	N/A	-	-
FL	134.6	GA	CA	3.4	VA	PA
GA	12.0	FL	IL	105.9	CA	NY
IL	63.4	CA	VA	47.3	TX	CA
IN	62.8	IL	FL	N/A	-	-
LA	12.7	FL	CA	N/A	-	-
MD	10.9	NJ	FL	N/A	-	-
MA	62.1	NY	CT	59.6	NY	FL
MI	103.0	CA	NY	N/A	-	-
NV	34.1	WA	CA	N/A	-	-
NJ	28.0	FL	MA	67.8	PA	NY
NY	85.2	PA	NJ	4.0	GA	VA
NC	20.7	FL	NY	43.6	FL	SC
OH	46.4	MI	FL	N/A	-	-
OK	51.6	CA	GA	N/A	-	-
OR	162.3	CA	WA	106.1	CA	WA
PA	45.5	VA	NY	46.2	NY	NJ
SC	10.8	MI	FL	N/A	-	-
TX	139.4	FL	CA	25.3	OK	IL
VA	15.6	OH	FL	36.6	NY	MD
WA	232.0	OR	CA	47.2	OR	CA
WI	160.5	CA	NY	N/A	-	-

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
2	6.2	CA	WA	0
4	0.0	WA	CA	1
15	0.7	IL	CA	0
16	6.7	CA	IL	0
17	0.6	CA	IL	0
23	0.0	CA	NY	1
24	2.9	VA	CA	0
26	2.7	TX	IL	0
27	6.3	NY	CA	0
28	7.6	IL	WI	0
29	4.1	IL	WI	0
34	0.7	CA	VA	0
35	3.7	IL	FL	0
37	0.8	NY	CA	0
40	4.2	GA	CA	0
44	1.2	FL	MI	0
45	2.1	FL	OH	0
46	5.5	OH	FL	0
49	1.8	NC	NY	0
50	0.0	NY	FL	1
51	0.0	NY	FL	1
52	6.4	NJ	PA	0
53	1.7	NJ	PA	0
54	0.0	NY	NJ	1
55	3.6	MA	NY	0
56	0.0	NY	CT	1
57	0.8	MA	NY	0
58	5.6	FL	MA	0
59	4.1	FL	MA	0

5.2.10 VE distance, angular resolution and critical features (P)

	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
AZ	11.8	CA	FL	N/A	-	-
CA	278.0	TX	CO	4.2	GA	AZ
CO	53.5	CA	NY	92.3	CA	TX
CT	36.9	MA	NY	N/A	-	-
FL	165.9	GA	CA	3.3	PA	VA
GA	29.4	FL	MI	103.8	CA	NY
IL	56.2	CA	VA	55.5	TX	CA
IN	69.0	FL	MI	N/A	-	-
LA	34.4	CA	FL	N/A	-	-
MD	15.4	FL	MA	N/A	-	-
MA	68.3	NY	CT	66.0	NY	FL
MI	73.0	CA	NY	N/A	-	-
NV	65.8	WA	CA	N/A	-	-
NJ	38.0	FL	MA	34.8	PA	NY
NY	61.0	PA	NJ	3.8	VA	NC
NC	13.1	FL	PA	14.0	NY	VA
OH	35.5	MI	FL	N/A	-	-
OK	33.5	VA	CA	N/A	-	-
OR	139.3	CA	WA	108.6	CA	WA
PA	33.8	GA	NY	57.2	NY	NJ
SC	7.8	FL	OH	N/A	-	-
TX	187.9	FL	CA	16.6	IL	LA
VA	12.0	FL	NY	25.6	FL	NC
WA	232.0	OR	CA	39.4	OR	CA
WI	127.2	CA	NY	N/A	-	-

	$d_e(v_*)$	ar_*
Smallest	7.8	3.3
Largest	278.0	108.6
Average	75.1	25.0
Median	53.5	37.1

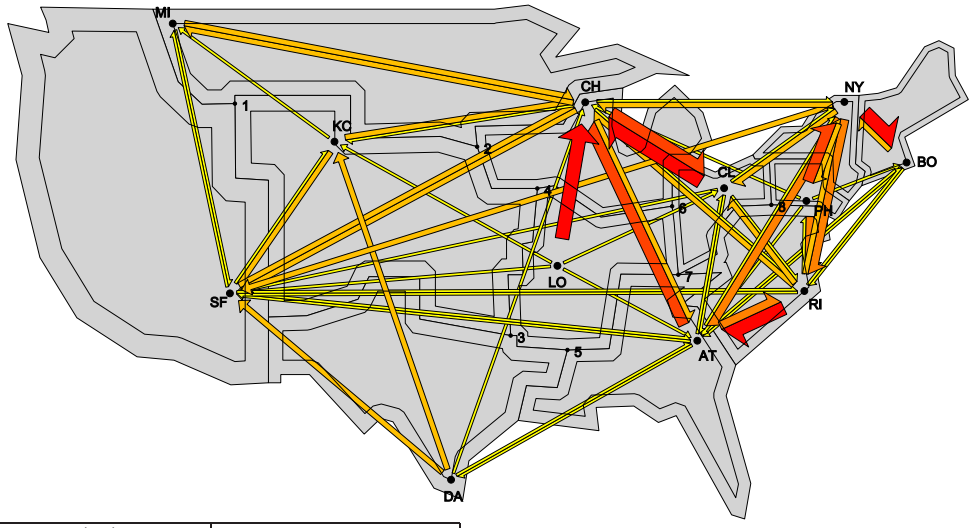
	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
3	7.0	CA	TX	0
5	0.0	CA	NY	1
9	0.2	CA	VA	0
18	3.6	NY	CA	0
29	3.8	IL	WI	0
31	8.0	CA	GA	0
35	7.0	IL	FL	0
42	4.0	FL	MI	0
43	2.3	FL	OH	0
44	1.5	MI	FL	0
47	7.0	FL	PA	0
48	1.7	NC	NY	0
53	7.9	NJ	PA	0
55	5.0	NY	MA	0
58	8.0	MA	NY	0

Figure 5.20: Figure 5.11 optimized for VE-distance, angular resolution and critical features (P).

5.3 Comparison to Tobler's flow map

Test results in this section use the same input as the flow mapping program by Tobler does. The input set is again the movement of 1 USD bank notes in America, but now map is divided into regions as defined by the 12 Federal Reserve Banks. To compare the results the final pictures are shown in Figures 5.22 and 5.23. For a fair comparison the input table from the actual mapper itself has been reused. The input set can be found in Table C.1 in Appendix C. Instead of showing all optimization we chose to show only the optimization for all three metrics with polygon regions (Figure 5.21).

5.3.1 VE distance, Angular Resolution and Critical Features (P)



	$d_e(v_i)$			ar_i		
v_i	dist.	u	w	angle	v_j	v_k
BO	83.8	NY	RI	14.6	RI	AT
NY	95.0	RI	BO	10.8	AT	PH
PH	19.6	NY	AT	40.5	RI	AT
CL	33.1	CH	PH	12.3	LO	SF
RI	56.7	BO	AT	11.5	CH	CL
AT	48.0	RI	SF	6.5	NY	RI
CH	118.8	SF	NY	8.2	LO	DA
LO	26.6	SF	RI	34.5	SF	KC
MI	210.7	KC	CH	37.2	CH	KC
KC	119.4	CH	MI	38.8	CH	LO
DA	251.3	AT	SF	27.9	SF	KC
SF	186.7	DA	KC	3.9	LO	RI

	$d_e(v_*)$	ar_*
Smallest	19.6	3.94
Largest	251.3	40.46
Average	103.9	20.56

	C-E distance			Overlap
cf_i	cf_d	u	w	cf_o
1	8.0	SF	MI	0
6	5.6	CL	SF	0
7	7.7	CH	AT	0
8	8.0	AT	NY	0

Figure 5.21: Movement of 1 USD bank notes between the 12 Federal Bank Reserve districts.

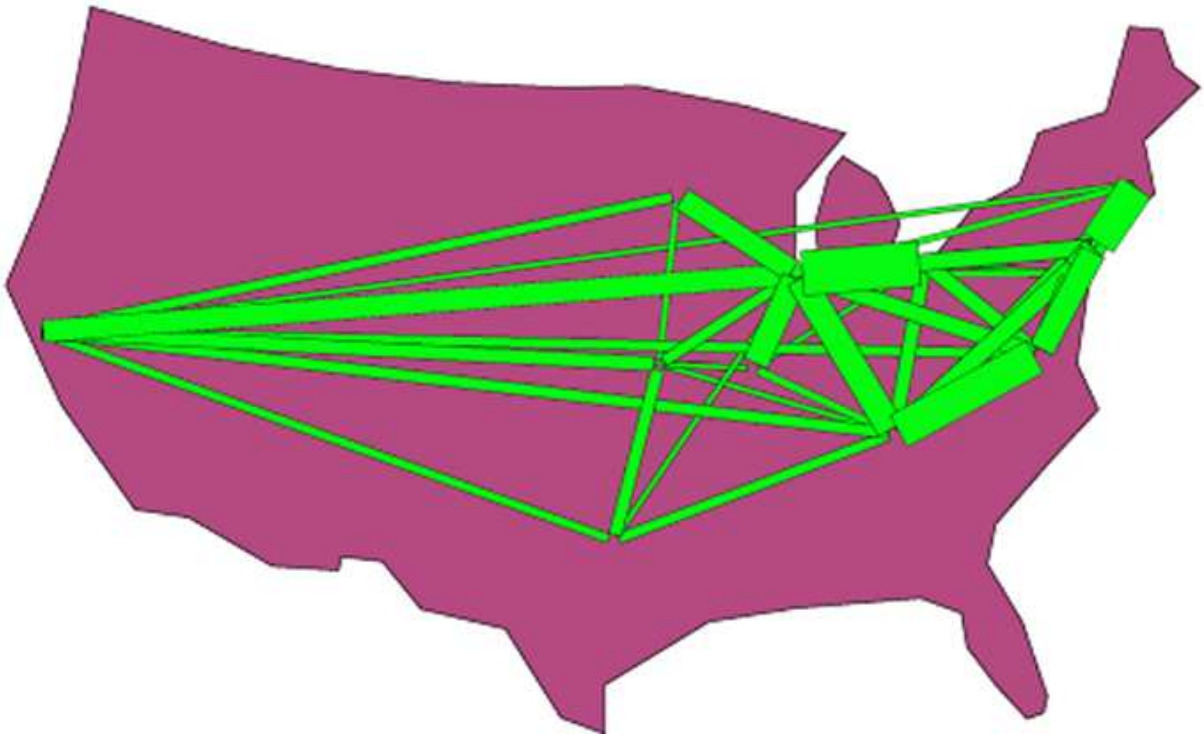


Figure 5.22: Data set from Table C.1 mapped by Tobler's flow mapper [29].

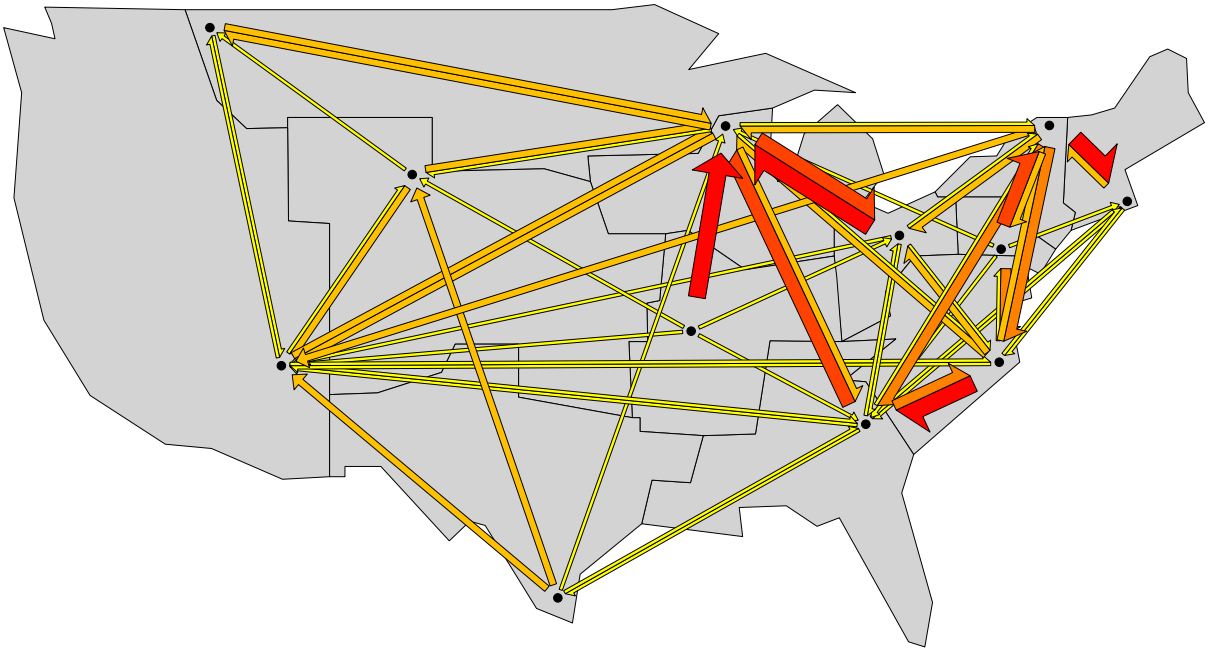


Figure 5.23: Data set from Table C.1 mapped by our implementation.

5.4 Discussion

The optimization of the maps are implemented as following. During each iteration each vertex is moved according to the methods as described in Section 4.3.3. If no better position can be found the vertex remains in its original location. Vertices are moved in an arbitrary but fixed order, first for vertex-edge distance and then for angular resolution. Once all vertices have been moved accordingly we try to improve the situation for the critical features, if required. This is done by moving the closest edge for each of feature on the map away for it. The distance the edge moves away from the edge depends on the shortest distance between the feature and the edge. The larger the distance the less the edge will move. For all movements it goes that the vertices can move in both x- and y-directions at the same time. During early stages of our research we also considered moving the vertices only by the x- or y-component in an alternating way. The maps generated by this method under the same circumstances as the normal optimizations always yielded less satisfying results. Hence we dropped this method of optimizing altogether.

Before going into detail for each test set it can be noted that because of the restriction certain areas on the map are always the bottleneck. For vertices in the center of the map it holds that optimizing vertex-edge distance can be difficult. A vertex near the boundary of the map only needs to move away from the center to get farther away from all the edges in the graph. For angular resolution the converse is true. Any vertex near the boundary of the map with more than a few incoming edges will yield the smallest angle in almost any case, whereas vertices located in the center of the map have a near-optimal value.

The optimization for vertex-edge distance by itself got stuck at local minima in certain situations, because vertices can only move away from edges and not the other way around. If we were to compute the contribution of all the forces exerted by the edges the algorithm would become much slower, therefore it was chosen to compromise this by moving only the edge (and thus its associated vertices) which is closest to any of the vertices. By repeating this process every iteration it improves the minimal vertex-edge distance metric significantly. It can however still get stuck in local minima for certain cases. Even though the vertices move around in the region in a discrete manner they can get stuck in non-optimal locations for the larger polygonal regions. This can be partly attributed to the fact that smaller regions have a smaller search space and therefore it is easier to optimize the vertex's locations. Also the less a vertex moves around the less its environment will be affected by the movement.

There is however more to say about the quality of the maps than comparing the metrics by themselves. Apart from the numbers the visual aspect itself is at least as significant. Therefore we split the discussion in separate parts, one for the metrics and one for visual appearance. Afterwards we give a complete overview.

5.4.1 Metrics

If we examine the criteria for the Dutch map with the data set considered in Section 5.1 shows that the map is reasonably dense and spread out evenly. The values for the smallest and average vertex-edge distance and angular resolution for all maps are shown in Table 5.1 for a quick overview. This test set has the property of generating the output as one would expect. Further experiments with other data sets, which we don't report in detail, confirm this behavior as described below. Optimizing for only vertex-edge distance naturally yields the best results for that metric. If we include the other metrics angular resolution goes up as well as the critical feature-edge distance, but this goes partly at the expense of vertex-edge distance. However because the updates occur only locally the differences are not very dramatic for circles and squares. The polygonal regions do exhibit significant differences as this type of region allows more movement for the vertices. This explains why the absolute differences are larger compared to the other region types. Purely based on the metrics for the minimum the optimization for all

three metrics with polygonal regions yield the best results. The same can be concluded when comparing the average values. The map as depicted in Figure 5.10 has the best appearance if we trade-off all three metrics.

Optimization type				Smallest		Average	
Region	VE.	Ang.	CF.	$d_e(v_*)$	ar_*	$d_e(v_*)$	ar_*
-				8.4	2.7	71.9	15.1
(C)	✓			31.1	6.3	95.9	16.0
(C)	✓	✓		29.5	6.8	93.2	17.4
(C)	✓	✓	✓	23.3	5.1	91.9	15.1
(S)	✓			29.9	5.9	84.8	14.7
(S)	✓	✓		27.3	6.6	86.6	18.2
(S)	✓	✓	✓	25.8	6.4	87.0	16.9
(P)	✓			*41.8*	5.7	*108.5*	18.5
(P)	✓	✓		26.8	*9.2*	93.5	*20.1*
(P)	✓	✓	✓	31.8	6.9	94.4	19.0

Table 5.1: Comparison table of the smallest and average values of the vertex-edge distance and angular resolution metrics for each map of the first test set used in Section 5.1.

For the American test set up in Section 5.2 the story goes quite differently. This data set has two extremely dense regions (namely Florida and California) and the edge from New York to Florida is wide enough to completely covers of the regions of the smaller states Maryland, North Carolina and Virginia. This area is clearly the biggest issue for this map. Next to that this map also has 59 critical features of which most are located near the already dense looking east coast. On the other hand 23 states on the map do not even have a single edge pointing to them because of the threshold applied to the data set.

Comparing the metrics as shown in Table 5.1 we notice that the differences are quite small apart from the trade-off between the decrease in the number of critical features-edge distance and vertex-edge distance. This can be observed best by comparing the maps of Figure 5.19 and Figure 5.20. The first one has 29 features within the 8 pixel range and the latter has 15 features within the same range, of which only 5 are at very close range (3 pixels or less).

That our optimization has some difficulties with certain local minima can be noticed if we compare the minimal vertex-edge distance of the maps with circular regions. Improving only vertex-edge distance resulted in a worse map compared to one which also includes angular resolution. Another noticeable difference is that the average value for angular resolution is worse for polygonal regions than for the other two region types. If we compare all values with equal weight, the polygonal regions are still, but only slightly, in the advantage. The best map based on the overall comparison of all three metrics is the map as shown in Figure 5.20, which is the polygonal region map optimized for all three metrics.

5.4.2 Visual appearance

For all settings it holds that the maps that are only optimized for vertex-edge distance tend to look more dense than the ones which improve other metrics as well. This makes sense as vertex-edge distance optimization increases the length of its the associated edges. If any of these edges is in highest classes, the edge will appear very dominantly on the map. When we include angular resolution, the result looks much more compact (e.g. just compare the situation of Limburg for Figures 5.8 and 5.9). As most maps have less vertices in the center than near the border optimization of the angular resolution metric implies that the vertices move inward towards the center.

Optimization type				Smallest		Average	
Region	VE.	Ang.	CF.	$d_e(v_*)$	ar_*	$d_e(v_*)$	ar_*
-				0.0	1.2	42.6	27.3
(C)	✓			8.5	2.9	69.4	27.3
(C)	✓	✓		9.9	3.3	68.1	27.4
(C)	✓	✓	✓	4.3	2.1	67.4	28.5
(S)	✓			8.0	2.4	65.8	28.1
(S)	✓	✓		8.5	2.6	67.0	*28.7*
(S)	✓	✓	✓	3.9	2.0	63.5	27.8
(P)	✓			*13.6*	3.3	*80.4*	27.4
(P)	✓	✓		10.8	*3.4*	78.6	26.8
(P)	✓	✓	✓	7.8	3.3	75.1	25.0

Table 5.2: Comparison table of the smallest values of the vertex-edge distance and angular resolution metrics for each map of the second test set used in Section 5.2.

Maps with worse values do seem to appear as being aesthetically more pleasant, when the situation around the thicker edges is better. From this we can conclude that the optimization of our flow maps could improve if we assign a higher priority to the thicker edges. Applying such a scheme leads to *weighted* optimization as we assign weights to our edges depending on their relevance. The optimization for critical features can contribute to an aesthetically more pleasing image, but because only 3-region points are identified as critical features in this implementation the effects seem less significant. This could also be due to the fact that most viewers spend most time looking at the edges to interpret the data. The base map becomes more important if the viewer is not immediately able to recognize the region.

5.4.3 Comparison to Tobler's mapper

The last comparison can be made between Tobler's mapper and our implementation. Both convey the same information and simplified the base map considerably, but result is very different. Tobler's map doesn't optimize or move the vertices on the thematic layer at all and therefore the edges between the vertices near the east coast are collinear just because the vertices where chosen to be positioned there. The result is the obfuscation of several edges making them hard to recognize. Secondly, the mapper doesn't show the interior region boundaries making it somewhat harder to judge to which area the edges point towards. Apart from the contents Tobler's mapper can only store images as a bitmap which can generate artifacts when the image is scaled.

Chapter 6

Conclusions

The research in this thesis has lead to several improvements for the problem of optimizing flow maps by automated means. By taking the base map into consideration when visualizing the edges the resulting flow map becomes more clear and better readable to the viewer. A few techniques have been introduced like the use of additional (dynamically) shorting of the edges to maintain the advantages the ascending thickness order without losing the arrow head visibility.

Given what has been done in this area of research so far the mapper created for this thesis is a step forwards, but there are still many possible additions worth implementing. The test results show that the approach taken in this thesis can be problematic for some types of graphs in certain cases such as dense areas. One could improve the situation by using the exact calculation of the optimal position for the worst vertex, which can be done for vertex-edge distance in polynomial time as shown in Section 3.4. Changing the order in which the vertices are optimized each iteration depending on the quality of the current situation or in a random order rather than a fixed arbitrary order may help too.

However as described in the discussion of the test sets in Section 5.4 some of the maps that are less optimal metrically-wise do appear as being aesthetically more pleasant. One situation for which this is true is that a good appearance of the thicker edges is more important than for the thinner edges. Therefore we could consider using a weighted optimization. Other than that there may be other metrics that we haven't discussed in this thesis which can help to further improve the appearance of the flow maps generated by automated means.

One metric which comes to mind here is minimizing the coverage of region borders. One could try to minimize the intersected area of the region borders by placing the edge such that it is at a straight angle with the line marking the region's border. Another thing to take into consideration is that we do not consider giving a penalty to vertices that ended up near the boundary of their respective regions. People associate the vertices more with the corresponding region if the vertex remains in the center (or the closest thing that can be defined as a center) of the region. This leads us to an important observation, which is that the human eye perceives data with a subjective view rather than with an objective one. Therefore the output of the automated mapper should always be verified and checked by people. They always have the last word about the quality of the map.

The use of edges that are displayed as polylines or curves will certainly allow improvements as they allow us to completely avoid problems that are otherwise imminent. This comes however at the price of making the optimization of the flow map a more complex task. For the polylines one might wish to restrict the number of angles and avoid using arbitrary angles for the edges (e.g. allow only vertical, horizontal and diagonal segments). Another possibility comes to mind if one allows the endpoints of the edges to roam freely inside their associated regions instead of

using a single vertex for all edges of a single region.

In all the difficulty of the problem even for smaller graphs makes it worthwhile to continue research on this topic, as *NP*-hardness can already be proven for the case where we are only interested in optimizing vertex-edge distance. It is however an open problem for two-dimensional vertices if this property still holds when the regions are restricted to 1 Dimensional line segments. There are many aspects of the base map and the thematic layer which require attention, and optimizing all of these criteria at the same time is not an easy task. The topic is far from exhaustively researched leaving open new techniques or ideas which is beneficial to the readability of the flow map in general.

Appendix A

Dutch flow map test set input table

This is the original collection of data on the migration of people for the twelve provinces of the Netherlands in 1996 retrieved from the website of the CBS [5] used for the flow maps of Section 5.1.

From	To											
	GR	FR	DR	OV	FL	GE	UT	NH	ZH	ZE	NB	LI
GR	-	2135	4586	1646	406	1165	1061	1902	1587	101	595	261
FR	2783	-	1256	1397	625	988	728	1826	1308	154	619	255
DR	4372	1095	-	2026	352	897	625	767	864	81	393	168
OV	1669	1044	1865	-	903	5238	1876	2111	2074	182	1236	449
FL	409	735	424	1154	-	1595	1245	4389	1312	153	579	225
GL	1310	909	936	5753	1444	-	6709	4084	4737	531	5530	1973
UT	662	688	635	1481	1420	6473	-	6933	5238	436	2595	771
NH	1253	1777	981	1817	6803	3920	7092	-	8684	521	2898	1137
ZH	1336	1356	1391	2167	1460	5782	6582	10124	-	2570	7478	1642
ZE	73	115	74	171	86	553	424	553	2035	-	2021	217
NB	597	447	376	1092	561	5856	3281	3359	6764	1918	-	4469
LI	235	148	151	470	193	2169	1161	1517	1837	212	4895	-

Table A.1: Data set of local movement of people between the 12 provinces of the Netherlands in 1996 [5].

Appendix B

US flow map test set input table

This is the original collection of data for the continental states of the USA with respect to the movement of 1 USD bank notes between each of the states. The data is retrieved input table from Tobler's mapper that comes along with the program [29].

From	To																							
	AZ	CA	CO	CT	FL	GA	IL	IN	LA	MD	MA	MI	NV	NJ	NY	NC	OH	OK	OR	PA	SC	TX	VA	WA	WI
AZ	-	92	31	2	17	9	15	7	3	4	4	11	23	4	10	9	10	8	16	7	3	38	9	21	7
CA	186	-	111	15	94	66	68	32	22	29	36	40	199	27	65	56	37	38	132	34	18	183	62	156	23
CO	35	56	-	3	22	11	14	6	4	5	6	8	11	4	9	10	9	12	11	7	4	47	12	18	7
CT	7	18	5	-	47	11	7	3	2	7	38	4	2	11	39	15	5	1	2	15	7	9	15	4	2
FL	22	65	26	18	-	157	38	28	17	24	30	39	15	35	70	96	47	10	6	44	38	81	58	16	13
GA	9	29	11	5	99	-	15	9	11	11	7	12	3	8	18	48	18	6	3	12	41	42	26	10	4
IL	48	74	27	6	86	35	-	85	6	9	11	43	18	12	21	21	30	8	7	15	10	64	19	13	81
IN	13	20	8	2	47	14	48	-	3	4	4	32	4	3	7	12	37	4	3	8	7	23	9	5	9
LA	4	18	6	1	22	20	6	4	-	6	2	4	3	2	5	9	5	6	1	3	4	86	9	5	2
MD	7	27	8	5	50	19	9	4	3	-	10	6	2	11	22	31	11	2	2	53	13	19	79	6	3
MA	10	44	8	29	68	13	10	3	2	10	-	6	3	14	48	14	8	1	3	17	5	14	15	7	3
MI	25	36	14	3	75	24	35	33	4	6	7	-	8	5	15	20	43	4	4	11	9	29	13	9	20
NV	19	60	10	1	8	3	5	2	2	1	1	3	-	2	4	3	4	3	10	2	2	12	4	11	2
NJ	12	35	8	16	119	27	12	5	3	24	25	7	7	-	98	37	11	2	2	110	17	21	37	5	3
NY	31	96	21	76	308	67	26	11	7	44	73	20	17	207	-	101	31	5	7	112	40	46	75	14	7
NC	9	27	9	5	58	51	12	9	7	18	8	11	3	10	20	-	19	6	3	18	65	33	63	8	4
OH	20	34	12	4	91	31	30	44	5	10	9	48	7	8	21	35	-	5	4	34	18	32	24	9	8
OK	9	17	10	1	12	8	7	4	5	3	1	4	3	1	4	6	5	-	3	3	3	83	6	6	2
OR	21	68	10	1	7	3	4	2	1	1	2	3	10	1	5	4	3	3	-	3	1	12	3	83	2
PA	17	39	12	11	92	27	16	11	3	51	20	14	6	88	67	43	43	3	4	-	21	25	47	8	6
SC	3	10	4	2	31	46	5	4	3	7	3	5	1	4	9	61	9	2	1	8	-	13	20	3	2
TX	45	116	63	6	77	59	39	22	57	16	13	28	18	11	28	43	27	73	12	19	16	-	39	31	13
VA	12	45	15	8	76	38	16	9	7	66	12	12	4	19	28	89	21	5	4	32	24	40	-	13	5
WA	38	95	18	3	19	11	9	5	4	6	5	7	14	3	10	8	7	5	72	6	4	31	11	-	6
WI	18	17	11	1	25	7	42	8	1	3	3	17	5	2	6	6	8	3	3	4	3	15	5	6	-

Table B.1: Data set of movement of 1 USD bank notes between the individual states in 1976 (x1000). The data was retrieved from the input tables included with Tobler's flow mapper [29]. To conserve space only the states after applying the average as threshold are shown.

Appendix C

Tobler's map test set input

This is the original collection of data for the twelve Federal bank districts in the USA with respect to the movement of 1 USD bank notes between each of the districts. The data is retrieved input table from Tobler's mapper that comes along with the program [29].

From	To											
	BO	NY	PH	CL	RI	AT	CH	ST	MI	KC	DA	SA
Boston	-	289	47	52	137	118	90	10	16	15	13	80
New York	602	-	231	209	388	307	286	15	48	26	18	261
Philadelphia	143	414	-	84	342	130	134	8	25	10	10	80
Cleveland	68	192	47	-	171	177	618	16	44	43	19	131
Richmond	150	266	158	226	-	578	295	20	62	54	22	152
Atlanta	122	159	57	186	319	-	439	30	51	78	102	189
Chicago	97	155	39	496	143	266	-	74	278	100	40	290
St. Louis	31	56	14	142	80	201	573	-	46	128	47	109
Minneapolis	14	26	11	32	29	41	295	10	-	51	14	138
Kansas City	20	41	8	55	40	71	215	33	129	-	86	247
Dallas	31	41	8	38	46	165	125	20	37	253	-	203
San Francisco	82	81	23	84	114	106	251	22	127	128	43	-

Table C.1: Data set of movement of 1 USD bank notes between the 12 Federal bank districts of the USA in 1976 (x1000) [29].

Bibliography

[1] http://www.eindhoven.com.

[2] Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. Computing envelopes in four dimensions with applications. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 348–358, New York, NY, USA, 1994. ACM Press.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[4] Sergio Cabello. *Geometric Problems in Cartographic Networks*. PhD thesis, University of Utrecht, department of Computer Science, Utrecht, NL, 2004.

[5] Centraal Bureau voor de Statistiek. http://www.cbs.nl, December 2006.

[6] J. S. Chang and Chee K. Yap. A polynomial solution for the potato-peeling problem. *Discrete and Computational Geometry*, 1(1):155–182, December 1986.

[7] Francis Y. Chin, Jack Snoeyink, and Cao An Wang. Finding the medial axis of a simple polygon in linear time. In *ISAAC '95: Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 382–391, London, UK, 1995. Springer-Verlag.

[8] Mirela Damian. Exact and approximation algorithms for computing optimal fat decompositions. *Computational Geometry*, 28(1):19–27, 5 2004.

[9] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

[10] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwartzkopf. *Computational Geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition, 2000.

[11] Borden D. Dent. *Cartography: Thematic Map Design*. McGraw-Hill, 5 edition, 1999.

[12] Christian A. Duncan, Alon Efrat, Stephen G. Kobourov, and Carola Wenk. Drawing with fat edges. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing: 9th International Symposium*, pages 162–177, Vienna, Austria, September 2001.

[13] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[14] Jirí Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Geometric systems of disjoint representatives. In *GD '02: Revised Papers from the 10th International Symposium on Graph Drawing*, pages 110–117, London, UK, 2002. Springer-Verlag.

[15] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software Pract. Experience*, 21(11):1129–1164, 1991.

[16] Robin Hartshorne. *Algebraic Geometry*. Springer-Verlag, New York-Berlin-Heidelberg, 1977.

[17] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[18] Chun Cheng Lin and Hsu-Chun Yen. A new force directed graph drawing method based on edge-edge repulsion. In *IV '05: Inproceedings of the ninth Internation Conference of Information Visualization*, pages 329–334, Washington, DC, USA, 2005. IEEE Computer Society.

[19] Mary J. Parks. American flow mapping: A survey of the flow maps in twentieth century geography textbooks, including a classification of the various flow map designs. Master's thesis, Georgia State University, department of Geography, Atlanta, GA, USA, 1987.

[20] Michael S. Paterson and F. Frances Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5(5):485–503, 1990.

[21] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *INFOVIS '05: Proceedings of the 2005 IEEE Symposium on Information Visualization*, volume 0, pages 219–224, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[22] Franco P. Preparata. The medial axis of a simple polygon. *Mathematical Foundations of Computer Science*, 53:443–450, 1977.

[23] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[24] Ernest G. Ravenstein. The birthplace of the people and the laws of migration. *The Geographical Magazine*, 3(2):173–177,201–206,229–233, 1876.

[25] Ernest G. Ravenstein. The laws of migration. *Journal of the Royal Statistical Society*, 52(2):241–305, June 1889.

[26] United States Postal Services. http://www.usps.com/ncsc/lookups/abbreviations.html#states.

[27] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, New York, NY, USA, 1996.

[28] Barbera Simons. A fast algorithm for single processor scheduling. In *Proceedings 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 246–252. Piscataway, 1978.

[29] Waldo R. Tobler. http://www.csiss.org/clearinghouse/flowmapper/.

[30] Waldo R. Tobler. A computer model simulation of urban growth in the detroit region. *Economic Geography*, 46(2):234–240, 1970.

[31] Waldo R. Tobler. Experiments in migration mapping by computer. *American Cartographer*, 14:155–163, 1987.

[32] Waldo R. Tobler. Global spatial analysis. *Computers, Environment and Urban Systems*, 26(8):493–500, November 2002.

[33] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphic Press, Cheshire, CT, USA, 2 edition, May 2001.

[34] Marc van Kreveld and Bettina Speckmann. On rectangular cartograms. *Computational Geometry: Theory and Applications*, 37(3):175–187, August 2007.

[35] Wolters-Noordhoff. *Grote Bos Atlas*. Wolters-Noordhoff B.V., 51 edition, 1995.

[36] Wolters-Noordhoff. *Grote Bos Atlas*. Wolters-Noordhoff B.V., 52 edition, 2003.