Enhancing Schematic Maps with Zones for Improved Route Planning

Master Thesis

R.P.J. van Happen

Supervisors:
dr. H.J. Haverkort
dr. M.A. Westenberg

Committee members:
dr. H.J. Haverkort
dr. M.A. Westenberg
dr.ir. H.M.M. van de Wetering

Eindhoven, November 2015
Abstract

Schematized maps are often used in order to simplify transit map networks. However, in order to do so the length of different routes often changes. This makes it hard to see which route would be the fastest route between a given source and destination. In order to remedy this problem and help users using the map, we divide the map into zones of approximately equal diameter (travel time-wise). Now a route that is shorter can be found as the route that traverses less of these zones. We present an automated approach to find and visualize such zones for schematized maps.
Preface

This master’s thesis is written as part of my graduation project in the Computer Science and Engineering master of Eindhoven University of Technology. This project was done internally in the Algorithms research group.

First and foremost, I would like to thank my supervisors Herman Haverkort and Michel Westenberg for providing me with guidance and support throughout the project. Special thanks also go out to Michel Cijsouw for thinking with me as a peer during the project. In addition, I would like to thank Huub van de Wetering for providing me with valuable feedback. I would also like to thank the members of the algorithms department for valuable information and inspiration during my education. In addition, I would like to thank my colleague students for supporting me during my education. Finally, My thanks go out to my parents, my brother and my sister, who supported me through the years.

Roel van Happen
October 2015
# Contents

1 Introduction 1
   1.1 Problem description ...................................................... 3
   1.2 Approach ........................................................................... 4
   1.3 Results .............................................................................. 4

2 Related Work 5
   2.1 Schematization ................................................................. 5
   2.2 Augmenting schematic maps ............................................... 5
   2.3 Set visualization ............................................................... 7

3 Finding a good zone distribution 11
   3.1 Problem specification .......................................................... 11
   3.2 Splitting up the problem ...................................................... 11
   3.3 Zones as an alternative measure .......................................... 12
   3.4 Criteria for a good distribution ........................................... 13
   3.5 Assumptions ...................................................................... 17
   3.6 Formalising the problem .................................................... 17
   3.7 Finding a good searching time ............................................. 19
      3.7.1 Testing triviality .......................................................... 20
   3.8 The solving process .......................................................... 20

4 Zone visualization 23
   4.1 Visual Criteria ..................................................................... 23
   4.2 Visualization using a Voronoi-based approach ......................... 24
      4.2.1 Voronoi diagram ........................................................... 25
      4.2.2 Merging Voronoi cells ..................................................... 25
      4.2.3 Trimming zones ............................................................. 25
      4.2.4 Drawing curved zones ...................................................... 26
      4.2.5 Creating space between zones ......................................... 26
   4.3 Visualization using an Energy-based approach .......................... 27
      4.3.1 The Bubble Sets algorithm .............................................. 28

5 Results 31
   5.1 Solving time ....................................................................... 31
   5.2 Visual results ..................................................................... 32
      5.2.1 Voronoi results ............................................................. 32
      5.2.2 BubbleSet results ............................................................ 33
   5.3 Finding a good zone size ..................................................... 49
Chapter 1

Introduction

The idea of schematizing maps has been around for a while. Harry Beck created the schematic map of London in 1931. The goal of schematizing the London underground was to reduce clutter and give underground travellers an overview of the underground network. Initially, there was one form of critique on Beck’s underground map: The schematic map did not show distances relative from one station to another. While this flaw did not stop the rise of schematic maps, it remains a flaw today. One major consequence of this is that because anyone looking at the map cannot know the distances, it becomes guesswork to decide which route is the shortest (fastest) to his destination. Figures 1.1 and 1.2 showcase the differences between schematized maps and geographic maps. The schematized map in Figure 1.1 is less cluttered around the center because the lines are more spread out over the map; however, many lines have been stretched or compressed compared to the geographic map in Figure 1.2.

Figure 1.1: One of the more recent schematic maps of the London underground made by Transport for London.
In order to more clearly illustrate the flaw of schematic maps, we zoom in on the maps given in Figure 1.1 and Figure 1.2. The zoomed-in versions are given in Figure 1.3a and Figure 1.3b respectively. Two stations are highlighted in these Figures: Notting Hill Gate and Baker Street. In the schematized map, both the route through Paddington and the route through Bond Street look equally long. In the geographic map however, the route through Paddington is clearly shorter than the route through Bond Street.
CHAPTER 1. INTRODUCTION

(a) A portion of the geographically laid out map of Figure 1.2 zoomed in. The shortest route between Notting Hill Gate and Baker Street clearly passes through Paddington.

(b) A portion of the schematic map of Figure 1.1 zoomed in. Both routes between Notting Hill Gate and Baker Street appear to be about equally long.

1.1 Problem description

In order to solve the problem of not seeing lengths on schematic maps, we want to augment schematic maps with information about the length of routes. The simple way of labelling all edges of the schematic map with their length does give all the information that is needed; however, in order to compare the length of different routes someone using the map has to add up the lengths of all the edges of both routes and compare the two numbers. Doing this for many different options requires a lot of effort and we seek to make it more intuitive to at least estimate the length of certain routes. To this end, this thesis introduces an alternative method of providing information about the length of routes. The core idea behind this alternative method is to draw a number of non-overlapping zones in the background of the map that together cover all stations on the map. Each of these zones represents a predetermined time to traverse. The number of zones a certain route then traverses can then be used as an estimate of the length of the route. This idea was originally proposed by Haverkort 13. Figure 1.4 shows an illustration of the zone concept applied to the Dutch railway network. A problem of this approach is that it is usually not possible to exactly represent the travel time on each route with a number of zones with a fixed travel time. A simple example is when you are using zones that represent 10 minutes to traverse, and there is one route in the network that takes 15 minutes to travel through. In this case you have to approximate the travelling time by either only letting the route pass through a single zone (an under-approximation), or letting the route pass through two zones (an over-approximation). It only gets more complicated as routes intersect, and each has to be approximated. The problem is now to find a distribution of zones that approximates the travel time for each route as good as possible.
1.2 Approach

The zones, as described in Section 1.1, can be drawn by hand by cartographers; however, we are interested in trying to generate a good set of zones automatically. In order to do this, we have split up the problem of finding good zones into two steps: The first step is to find a distribution of stations over zones such that the zones on each route are a good estimation of its length. How this is done is described in more detail in Chapter 3. The second step is to visualize the distribution we found in the first step and make the zones look natural. For this visualization, we have analysed two different approaches. The first approach is based on Voronoi diagrams and is described in Section 4.2. The second approach is based on potential energy. This approach is described in more detail in Section 4.3. We have tested both approaches and compared their results in Section 5.

1.3 Results

We have compared both visualization approaches. While the Voronoi-based approach does guarantee that no overlap occurs, the zones sometimes end up having very sharp angles. The energy-based method however creates more natural zones, but overlap may occur between different zones.
Chapter 2

Related Work

2.1 Schematization

The problem of drawing schematic maps has been researched extensively[14][18][24]. These approaches have been compared by Wolff[27]. In addition to these schematization methods, Cabello et al.[5] has also investigated automated schematization. However, regardless of the schematization approach used, schematic maps often distort the original map in such a way that the shortest route in the schematic map no longer corresponds to the shortest route in the original map. One approach by Milea et al.[16] describes an approach based on the approach by Wolff[27]. This approach adjusts the schematization from Wolff in order to ensure that the shortest routes on the original map remain the shortest routes on the schematic map. Sadly the results of their test code were inconclusive. The approach does look promising and this thesis uses a similar iterative approach in finding a good solution.

2.2 Augmenting schematic maps

For augmenting schematic maps, Stevens[23] proposes to use markers. These markers each represent a certain travel time. By counting them, the travel time of a route can be found. However, replacing the number representing the length of a route with a number of markers representing the length may not help too much for finding the fastest route. In both cases counting and adding up the travel time of individual segments is required.

Haverkort[13] discusses several methods to augment a schematic map with as goal to convey information about travel times to users.

**Thickness:** The first of these methods is to adjust the line width of connections depending on how fast the connections are. This means that faster connections will be drawn more bold, to attract attention. This approach might be effective. Because Cijsouw[7] was already investigating this approach at the time of writing, we did not further investigate this approach.

**Wiggled lines:** The second approach is to draw slower connections as more wiggled lines. This means slower connections are drawn as longer, twisted lines. While the idea seems straightforward, an unwanted effect might be that the wiggled, slower lines attract more attention from the viewer. This makes it harder to spot faster connections.

**Markers:** Another approach is similar to the last one; serpentine-like markers are placed on each connection based on how slow this connection is compared to the length of the line that is drawn. For example, a marker is placed for every 5 minutes; now one connection takes 15 minutes longer to traverse than another connection, when both connections are drawn equally long. The former connection will now have 3 more markers on it than the latter.
CHAPTER 2. RELATED WORK

connection. In theory this is a good idea, but the main disadvantage might be that, like the last approach, routes with more markers attract more attention, while you want to focus attention on routes with less markers.

Heat map: It is also possible to draw a heat map in the background of the map. On this heat map, bright colors correspond to a faster travelling speeds and dark colors correspond to slower travelling speeds. The problem with this approach is that the part of the heat map that is important to determine how long a route takes is obscured by the route itself drawn over it. While it is still perfectly possible to see how fast a route is by looking at the heat map right next to the route, we feel that this does in fact detract from the effectiveness of the approach.

Transformed grid: Another approach that Haverkort proposes is to create a grid covering the map with all stations in their geographical locations. When schematizing this map, the grid is transformed the same way the stations are moved to create the schematic map. The result is that the grid is stretched where travel is fast and compressed where travelling is slower.

Subdivision: Haverkort also proposes an approach where the schematic map is subdivided in regions. The idea is that each region takes a certain time (e.g. 20 minutes) to traverse. The biggest problem of this approach is that the number of regions a route can traverse is discrete. If a route travels into a region, it can be hard to judge how long the journey in that region might take, especially if that region is not exited.

Bloblike zones: The last approach that Haverkort proposes is based on the subdivision-based approach. Instead of directly drawing the subdivision, each region is drawn as a blob-shaped zone with a smooth boundary. While this is more complicated than the previous subdivision-based approach, it is also aesthetically more pleasing. This approach is the approach sketched in Figure 1.4.

We take a more in-depth look at the zone-based approach from this paper; in particular, we look at a method to find such zones automatically. We hope that despite the discreteness problem that comes with the idea of zones, zones can be used to help route planning.

As already mentioned in the line thickness approach from Haverkort, Cijsouw\[7\] has researched how line thickness in schematic maps can be modified to guide users towards faster routes. In addition to this, Cijsouw looked into maps that are optimized for route planning where all routes that are considered have the same source station. Such a map can for example be physically distributed in that station to help travellers there get to places elsewhere. In these 'source-based' maps, he researched several other modifications as well. The first of these approaches is to use dashed lines for connections that are not part of any fastest routes from the source station to any destination. This hints travellers coming from the source station to avoid these connections. Similarly, he researched adding arrows on these source-based maps that indicate the direction of travel along routes from the source station. An example illustrating the combination of adapted line thickness and dashed lines with the source based approach is given in Figure 2.1

Cijsouw has also researched several adaptations of the schematization algorithm from Wolff\[27\]. The first adaptation he has researched is reducing line bends on connections that are more frequently used. The second adaptation he has researched is adapting the schematization to create more space on the map for areas with many lines. Cijsouw organized a user study to test the success of his modifications. While the results from this user study look promising, they were statistically inconclusive.
CHAPTER 2. RELATED WORK

2.3 Set visualization

In addition to approaches related to schematic maps, we are also interested in visualizing any zones that we have found in an attractive manner. The problem of visualizing zones each containing subsets of a metro map is quite similar to the problem of visualizing sets. Therefore we have also looked at effective methods to visualize sets on a given embedding.

The more traditional way of visualizing multiple sets is by using Venn or Euler diagrams. A description of properties of Euler diagrams is given by Stapleton et al.[22]. However, the concept of zones relies on the stations having a fixed position. This does not mesh well with Euler diagrams, since simple shapes do not always suffice without moving the stations around.

Therefore we have looked into visualization methods that can handle arbitrary data without having to move any items. One approach by Collins et al. allows for this. They designed the BubbleSets[8] approach. BubbleSets visualizes each set of items by computing the area that is close to one or more items of the set. This is done by using an energy-based approach. In this approach, a certain area has more affinity (or positive energy) for a set of items the closer it is to an item in the set. An area has less affinity for a set of items if it is close to an item that is part of another set. Now the final affinity of an area is computed as a sum of all the positive affinity...
CHAPTER 2. RELATED WORK

contributed by items of the set nearby minus the sum of all the negative affinity contributed by items of other sets nearby. Each set is then drawn by enclosing the area where the net affinity is greater than some threshold. This approach can deal with overlapping sets quite well, but it is also perfectly suitable for non-overlapping sets. For this reason, we decided to adapt the method for visualizing our results. An example of BubbleSets is given in Figure 2.2.

Another visualization approach uses lines to represent sets. This approach has been researched as LineSets[2]. The advantage of using lines is that they take less space than using areas. This is helpful when much overlap occurs, because there will be less clutter. However, we seek to avoid overlap altogether. For this reason we prefer to use areas because they better convey which parts belong to which set.

Another set visualization method is KelpFusion[15]. Visually, KelpFusion is a bit of a mix between LineSets and BubbleSets. First, KelpFusion connects all the points in a set similar to LineSets. Instead of only connecting all the points in each set with a spanning tree, KelpFusion may add extra lines between points that are geographically much closer than through the tree that is already made. Now whenever appropriate, it fills faces enclosed by connecting lines. With the most extreme settings, KelpFusion will draw a set as a convex hull of the points it contains if there are no points inside that are not part of the set. An example of how the KelpFusion visualization looks is in Figure 2.3. However, we believe that KelpFusion is not suited for displaying zones. This is mainly because KelpFusion does not try to avoid overlapping multiple sets whatsoever. While KelpFusion does avoid filling a face for a certain set if there is a point inside that face that does not belong to that set, this does not avoid all overlap. This is because KelpFusion does not restrain itself from drawing connecting lines through other sets. While this is not a problem if only convex zones are involved, sometimes using concave zones might be necessary. For this reason we believe that a BubbleSet-based approach will be more suited, as its approach allows for a repulsive effect between zones that limits overlap.
Figure 2.2: An example of BubbleSets used by the authors to visualize three different types of venues in lower Manhattan.
Figure 2.3: The KelpFusion approach used by the authors to visualize three different types of locations in New York.
Chapter 3

Finding a good zone distribution

3.1 Problem specification

Before we analyse possible solutions, we will properly define the problem. We are given a (schematized) map represented by a graph \((V,E)\) drawn on the plane. Each edge has an associated length, which represents the travel time needed to traverse it. We assume that there is no other time required while travelling apart from the travelling time between nodes on the map. In particular, we do not account for time lost when switching between different lines. In order to simplify visualization, we assume the graph itself is planar. Each node \(n \in V\) has coordinates \(x\) and \(y\) that indicate its location when drawn on the plane. For the purpose of simplifying the visualization, we assume that each edge \(e \in E\) is drawn as a straight line that connects both of its endpoints.

3.2 Splitting up the problem

In order to create a good set of zones to draw on the map, we separate the problem into two parts. We initially ignore differences between zone distributions where the number of zones each path crosses is the same, so we focus on which nodes are in which zones. This comes down to finding out how many zones are needed and on which edges the boundaries lie for each zone. We will call this step the ‘zone finding’ step. We describe how to find a good solution for this sub-problem in this chapter.

Now the second sub-problem is how to draw the zones themselves on the map when we have already decided on their placement in the previous step. We will describe this step in Section 4. Figure 3.1 shows an example of the two steps we split the problem into.
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

Figure 3.1: An example of how the problem of finding zones is split up. In the zone finding step, we decide that the three points in the center should be together in one zone. If we now want to draw a zone surrounding these points we want the outline of the zone to be in the grey area, crossing the red edges. In the visualizing step we try and find a nice shape for the zone.

3.3 Zones as an alternative measure

The core problem is the ‘zone-finding’ step as described above; that we want to find a good distribution of zones over the graph such that each route traverses an appropriate number of zones. This distribution is considered to be a partition of the plane the graph is drawn onto. An example of a partition on a graph is given in Figure 3.2.

Based on such a partition, we can come up with a new length measure to measure the length of paths. This new length measure, zone length, is defined in Definition 1.

From such a partition we can then count the number of times an edge leaves a zone and enters another zone. This amount is used to define a new distance metric to determine the distance between two nodes in Definition 2.

**Definition 1.** The zone length of a path between two nodes $A$ and $B$ is equal to the number of times one zone is exited in order to enter another on the path between $A$ and $B$.

**Definition 2.** The zone distance between any two nodes $A$ and $B$ is equal to the minimum zone length of any path between $A$ and $B$.

The goal, as mentioned in Chapter 1.1, is to create zones that each have approximately the same size. This way, we can use the zones to compare the length of routes. We will call this desired size the zone size.
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

Figure 3.2: A partition on a simple graph. In this example the edge between E and D crosses from one zone to another twice and the edge between D and C crosses no zone boundaries.

3.4 Criteria for a good distribution

The focus of this research is to automatically find a good partition of the route graph in zones. There are several criteria that affect the quality of a good partition. Below we list the criteria that we consider important.

1) It is crucial to the concept of zones that traversing a zone takes time proportional to the zone size. This can be more formally defined as follows: When we have a path that traverses \(X\) zones, we want the travel time to approximate \(X \times Z\), where \(Z\) is the desired zone size.

2) There is no path between two nodes that, due to the zone partition, appears faster than the shortest path between those two nodes. More formally, if \(A\) is the shortest path between two nodes, then there is no path \(B\) such that path \(B\) traverses strictly less zones than \(A\). This is because we do not want to mislead users into using a path that, due to the partition in zones, seems shorter than the actual shortest path. Preferably, we would want paths that are longer than the shortest path to also traverse more zones so users can more easily pick out the shortest path. An example of when there can be confusion due to a longer path appearing shorter is given in Figure 3.4.

3) For usability, it is important that the number of zones that any given path passes through is easily observable and countable. This means that it should be clear when a path traverses a certain zone and when it does not. In addition, we want users to be able to easily count the number of zones traversed without miscounting. For this it is important that individual zones are not too small. An example of when zones are too small is given in Figure 3.5.

4) Different paths between two nodes should preferably travel through different zones. We want to use zones to be able to differentiate between different paths, so users can see the differences between different paths. A very bad example that we want to avoid is to have the entire network contained in a single zone. In this scenario, the zone cannot be used to say...
anything about which path is shorter because no path actually leaves this zone. Therefore, each path has the same zone length. An example with only one zone is given in Figure 3.6.

These last two criteria are at odds with one another. Criterion 3) states that there should not be too many zones on a path while criterion 4) states that there should not be too few. Therefore, in order to satisfy both criteria a compromise should be made. This compromise is finding a suitable zone size such that there are not too many zones, nor too few. It is not clear upfront what the best number of zones is. We suspect that the optimal number of zones actually varies depending on the structure of the map; therefore, we leave this as a parameter in our approach. Instead of directly introducing a parameter for the number of zones, the number of zones will be determined by the zone size, which we will use as a parameter. We researched into finding a suitable zone size for one of the maps we used, the results of this are given in Section 5.3.
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

Figure 3.4: With this zone distribution, there are less zones between D and E if you travel from D via B and A to E than the direct path between D and E. If this map would be geographically accurate, this can confuse people into taking the detour via B and A.

Figure 3.5: If there are many zones on every path, it requires a lot of counting to figure out which path has a shorter zone length and is therefore shorter.
Figure 3.6: If many paths have the same zone length, the different paths become indistinguishable. With very few zones, this occurs often.
3.5 Assumptions

In order to simplify the problem, we make several assumptions.

- First, we assume that the network is an undirected graph. We make this assumption, because a directed graph can create problems. The most important problem occurs when the weight of an edge from node $A$ to node $B$ is different from the weight of the edge from $B$ to $A$. In this scenario a path from $A$ to $B$ is longer or shorter than a path from $B$ to $A$. If the length difference is large enough, it can be impossible to create the right number of zones on the path between $A$ and $B$. This is a reasonable assumption to make because nearly all connections in a transport network are used in both directions.

- In order to simplify visualization, we assume that the network does not contain parallel edges and is thus not a multigraph. While it is technically possible to draw an extra zone on one edge that is parallel to a shorter edge, this is very complicated. In particular, finding a way to automatically visualize such a scenario is hard. Despite making this assumption, we can still handle multigraphs by simplifying them beforehand. Any parallel edges can be eliminated by removing all of them except the one that has the shortest travel time.

- We assume the graph is planar. This is in order to avoid scenarios where two zones are forced to overlap with each other.

3.6 Formalising the problem

In order to find a good partition in zones, we convert the problem to a MIP (Mixed Integer Program[19]). As mentioned in Section 3.5, we decided to consider the zone size an input parameter.

The first step in this process is to expand the graph into a complete graph. For each edge between nodes $A$ and $B$ that has to be added in order to create a complete graph, its length is set to be equal to the total length of the shortest path between $A$ and $B$. An example of such a construction is shown in Figure 3.7a.

Adding variables

Now for each edge in the clique, there is one attribute that represents the zone length of this edge. The value for this attribute is equal to the zone length between both endpoints of the edge in the current solution. Because a negative zone length does not make sense, we demand that these attributes are non-negative. Consistency constraints are needed to define the number of zone crossings of the newly added edges. For each direct edge that was added in the construction of the clique, the following constraint is added: The zone length of this direct path between $A$ and $C$ is equal to the sum of the zone lengths of the edges from the shortest path between $A$ and $C$. So if a direct edge $AC$ is created using the edges $AB$ and $BC$, then the zone length $AC$, denoted by $c(AC)$, should always be equal to $c(AB) + c(BC)$. These constraints we add on the zone length for the edges added in Figure 3.7a are displayed in Figure 3.7b. With these constraints, the zone length of the shortest path from one node to another is uniquely defined by the zone length of each edge in the original graph.

The optimization criteria

We can now use the attributes we just defined to define our optimization criteria. In the optimization criteria, we want to minimize the difference between the travel time of a route and the travel time implied by the zone length of it. We do this using the direct edges we added earlier; for each pair of nodes $A$ and $B$ we add the following term to the optimization criteria: If the zone length between the pair is equal to $c(AB)$ and the total distance equal to $d(AB)$,


CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

(a) The dashed lines show new edges that are added between A and all other points. Since the edge AB is already present, it is not added. For all added edges, the length of the newly added edge is equal to the length of the shortest path between both endpoints.

\[ l(AC) = l(AB) + l(BC) \]

\[ l(AD) = l(AB) + l(BC) + l(CD) \]

(b) For the direct paths AC and AD the consistency constraints \( c(AC) = c(AB) + c(BC) \) and \( c(AD) = c(AB) + c(BC) + c(CD) \) are added.

then we add the term \( (c(AB) - d(AB)/Z)^2 \). We decided to use a sum of squared terms in order to more heavily weigh paths where the zone length times the zone size differs largely from the length of the path. Now the goal is to minimize the sum of all these quadratic terms. Because the objective function is a sum of squares, it is convex. This means that the objective function is non-negative for all possible combinations of variable values. Because the objective function is convex and the constraints are linear, the MIP is a convex program. There has been a lot of research on convex programs and multiple effective algorithms are available[25]. Even though the MIP is still NP-hard, heuristics from convex programs can be applied by the solver to speed up the search.

Adding extra constraints

In order to solve this MIP we use IBM ILOG CPLEX Optimization studio. The manual for CPLEX Optimization Studio is found in [9]. In essence, we solve this MIP to find a good solution; however, when using only these constraints an unwanted scenario might occur. Recall that in Section 3.4, four criteria were given for a good partition. If only the constraints described above are used, we found that it often occurred that a longer path ended with a lower zone length than the shortest path, which would conflict with criterion 3). This can occur because for one path it might be hard to accurately draw the correct number of zones without detracting from other paths. In such a case, a compromise is often made by the solver in the solution to deviate from the most optimal number of zones on this route. As a result, it might occur that this route ends up with a shorter zone length than another route while that other route has a shorter length. This situation is shown in Figure 3.4. We want to completely disallow this situation. Definition 3 defines such a situation as a Conflicting solution.

Definition 3. A solution to the MIP is conflicting if (and only if) there is a path between any two nodes a and b that has a strictly shorter zone length than the shortest (normal) length path between a and b.

In order to disallow these conflicts, we introduce additional constraints that state that there cannot be a path between two nodes A and B that has a shorter zone length than the shortest
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

path between $A$ and $B$. If we were to introduce such constraints for every pair of nodes and all paths that could be shorter than the shortest path, it would bog down the MIP tremendously. This is because in the worst case, there are $V^4$ pairs of paths that need to be checked, where $V$ is the number of vertices in the graph.

Instead of introducing constraints for all pairs of paths, we first solve the MIP without any of these additional constraints until we find a decent solution. When we do find a decent solution, we check for all pairs of nodes if there is a path shorter than the path with the lowest zone length between that pair of nodes; for this we use Dijkstra’s algorithm[17] twice, using two different distance measures. The first distance measure is the travel time over a certain path from one node to the other. The second distance measure is the zone distance measure defined previously. We find the shortest path for both distance measures and if the shortest path found with the zone distance measure has a strictly shorter zone length than the path with the shortest real distance, we found a conflict and add constraints for this pair of paths as mentioned above. After all pairs of nodes are checked like this, we restart the solver with the expanded MIP. This process is repeated until we find a solution we are satisfied with and that does not have any such conflicts.

3.7 Finding a good searching time

Short runs and Long runs

One problem remains; the problem of determining how long to run the solver before we consider the found solution satisfactory. Since the running time of the solver is exponential in the number of nodes, we cannot expect to run to completion in a reasonable amount of time. We want to keep the solving time as low as possible for any runs in which we expect to find conflicts to add to the MIP, as any solution found in these will likely not be used as the final result. However, enough time is needed for solving, otherwise the solver will not find any solution other than the trivial solution. The trivial solution in this case is that all nodes are part of the same zone, therefore the zone length of any path is always 0. In this solution, no conflicts are possible, because a path with a zone length shorter than 0 cannot exist.

In addition, we also want a solution of good quality in the end. In order to judge quality, we use the first criterion from Section 3.4: The zone length of each route should be as close as possible to representing its travelling length. If we only use a single searching time for the solver, we have to choose between fast runs and a good quality solution. To this end, we decided to use two different solving times $t_{short}$ and $t_{long}$. Here $t_{short}$ is the solving time we use when we try to find a solution that has conflicts. We will use $t_{short}$ as our stopping criterion when the previous solution contained one or more conflicts. $t_{long}$ is the solving time we use to try and find a good solution that will produce a good zone partition. We use $t_{long}$ as our stopping criterion when the previous solution did not contain any conflicts. However, there is one exception: When we only manage to find the trivial solution, we double the time $t_{short}$ and run the next run using $t_{short}$.

Short runs

Now we need to find good solving times for $t_{short}$ and $t_{long}$. First we discuss $t_{short}$: We find $t_{short}$ by using exponential search. Initially, we start with a really short search time $t_0$ as $t_{short}$. The exact value of $t_0$ is not important, important is that $t_0$ should be at most about as long as the time required to find a solution with conflicts. Now we run the solver for time $t_{short}$ ($t_0$ initially). After running for this time, we check if the found solution is nontrivial. How this is done exactly is described in Section 3.7.1. Now if we did not find a nontrivial solution, we double the searching time $t_{short}$ and solve again. We keep doubling $t_{short}$ like this until we find a nontrivial solution. When we do find such a nontrivial solution we can analyse this solution to find any conflicts. If we do find conflicts, we need to solve the new MIP. Important is that now the MIP
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

is more complex, so it will likely take longer to find a nontrivial solution. Because of this, we
need to check again if the found solution is nontrivial and increase the search time $t_{short}$ if need
be. This process will ensure that $t_{short}$ is never more than twice the minimum needed solving time.

Long runs

We also need to find a good value for the solving time $t_{long}$; this is the time that we spend
solving in order to find a ‘good’ solution. We have tested two different methods for determining
$t_{long}$: The first method is to directly relate $t_{long}$ to $t_{short}$. We decided to use a linear function
$t_{long} = 30 \times t_{short}$. We chose for this simple function because finding out which functions work
well and which ones do not requires extensive testing. The second method is to use a measure used
in optimizing: the optimality gap. The optimality gap is an upper bound on the improvement
left for the objective value of the MIP (As described in Section 3.6). It represents the difference
between the best objective value found so far and the current known lower bound on the objective
value. Formally, the optimality gap is equal to $(LB - opt)/opt$, where $LB$ is the found lower bound
and $opt$ is the objective value of the best found solution. The optimality gap decreases in two
ways while solving: First, the optimality gap decreases as the current best solution is improved.
Second, the optimality gap decreases when the lower bound of the optimal solution is increased by
finding a tighter bound. For this method we ran the optimizer until the optimality gap threshold
was reached instead of using $t_{long}$.

We tested both these methods and concluded that the time-based method is more consistent.
From the tests we also derived that the value 30 worked well as the factor between $t_{long}$ and $t_{short}$.
The test results we used are given in Section 5.1.

3.7.1 Testing triviality

When we solve the MIP, we want to find a solution other than the trivial solution. This means
that, whenever we end our search with a solution, we need to check if this is indeed a solution
other than a trivial solution. We do this by comparing the zone size to the diameter of the largest
zone. We define this maximum zone diameter as the longest travelling distance between two nodes
that are in the same zone. We consider a solution trivial if the maximum zone diameter is longer
than $x \times zone$ size. Here $x$ is a fixed integer. In our tests, we used a value of 5 for $x$. For our
tests, this value was low enough to ensure that the graph was subdivided in multiple zones. This
gives us a solution that is sufficiently complicated to analyse. The value is also high enough to
only refuse obviously oversized zones in the solution.

3.8 The solving process

In the end, the solving process consists of a three-step process. The first step is to solve the MIP
for time $t_{short}$ and check if the solution is trivial to see if this time needs to be increased. The
second step is to check the solution found in the first step for any conflicts. If any conflicts are
found, they are added to the MIP and the entire process is repeated. If no conflicts are found,
the third step is to extend upon the solving done in the first step by increasing the solving time
to $t_{long}$. If there are still no conflicts found, then we are done with the solving process; if there
are conflicts found, they should be added like in the second step, and the solving process should
be repeated. This solving process is summarized by the pseudocode below. In this code, if no
conflicts are found (i.e. $conflicts = \emptyset$), then the variable $conflicting$ is false, otherwise $conflicting$
is true.

Algorithm restartSolve
1. $conflicts \leftarrow \emptyset$
CHAPTER 3. FINDING A GOOD ZONE DISTRIBUTION

2. \( t_{short} \leftarrow t_0 \)
3. repeat
4. \hspace{1em} repeat
5. \hspace{2em} repeat addConflicts(conflicts)
6. \hspace{3em} solve for time \( t_{short} \)
7. \hspace{2em} conflicts \leftarrow \text{findConflicts()}
8. \hspace{1em} \textbf{until} \ (\neg \text{conflicting})
9. \hspace{1em} sufficient \leftarrow \text{true}
10. if \ \neg(\text{zones sufficiently small})
11. \hspace{1em} \textbf{then} \ \( t_{short} \leftarrow 2t_{short} \)
12. \hspace{1em} \hspace{1em} sufficient \leftarrow \text{false}
13. \hspace{1em} \textbf{until} \ (\text{sufficient})
14. solve for longer time \( t_{long} = 30t_{short} \)
15. \hspace{1em} conflicts \leftarrow \text{findConflicts()}
16. \hspace{1em} \textbf{until} \ (\neg \text{conflicting})
17. \hspace{1em} \text{Process results}
Chapter 4

Zone visualization

4.1 Visual Criteria

In this section we define several criteria that our visualization should satisfy:

- First, we want the zones to be drawn as a smooth shape. The curvy aesthetic is what makes the zones look good.

- Second, we want any zones created to be drawn as circular as possible. This is because we do not want to have any long and thin zones if these can be avoided. Long and thin zones can be confusing because they can take up space that is disproportionally larger (or smaller) than the area that is travelled through, depending on the direction of travel. If such a zone is traversed in the direction in which it is stretched, then the route travelling along that direction seems longer than it is. This is because there may be more zones close to the route, however less of these zones are actually travelled through because the route is travelling through the stretched zone for a longer length. On the other hand, if such a zone is traversed in the direction in which it is compressed, then the route travelling along that direction seems shorter than it is. This is because the stretched zone only covers a smaller stretch of the route, while still being pretty large. As a result, there are more large zones on this route, making it seem faster. An example of why non-circular zones can be confusing is shown in Figures 4.1a and 4.1b. Non-convex zones can be even more confusing in bad cases, so we would prefer to avoid them. We also believe that circular zones are visually more pleasing than stretched or non-convex zones.

- Third, we do not want zones to overlap anywhere. This might be pretty straightforward, but if multiple zones overlap these zones are possibly interpreted as only a single zone while they are in fact separate. In addition, it can be harder to see when a route travels through only one of these overlapping zones or through both. For the same reason we also want to keep sufficient space between zones.

- Fourth, we want each zone to contain the nodes/edges that are actually part of it (and no other zones/edges). This is to avoid confusion of when a route passes through a zone. An example of a confusing situation that might be created when a node is not drawn inside a zone it is part of is given in Figure 4.2. In order to make it more clear that a node is contained in a zone, we want to keep some space between each contained node and the boundary of the zone.
(a) Two routes passing through three elongated zones. Route A only passes through one zone while route B passes through all three. The elongated zones create the illusion that both routes are about equally long, while they are not.

(b) The same two routes as in Figure (a), this time passing through three circular zones. It is more clear that route B travels through more zones than route A.

Figure 4.1

Figure 4.2: Node X is not contained in the drawn zone. If it would be part of the drawn zone in the underlying representation, the route between A and B will seem shorter than it actually is. This is because it does not pass through the zone in the visualization while in reality it would.

4.2 Visualization using a Voronoi-based approach

The first approach we have used for visualizing our zones is based on Voronoi diagrams. Aurenhammer[3] provided a survey on Voronoi diagrams. The idea behind this is that a Voronoi diagram is constructed based on the nodes of the route graph. After this diagram is constructed, the Voronoi cells of all nodes that belong to the same zone are merged into one cell. This cell represents this zone in a polygonal format. In order to limit the drawing area of the zone to the area surrounding the nodes in it, we limit the cell by intersecting it with an enlarged convex hull of the nodes. Now a curvy zone is drawn based on this intersection.

In Section 4.2.1, we explain the exact Voronoi algorithm we use. In Section 4.2.2, we explain the details of combining all Voronoi cells that belong to the same zone. In Section 4.2.3, we explain how we determine the intersection of these combined cells and the convex hull. In Section 4.2.4, we explain how we draw curved zones based on the intersection. Finally, in Section 4.2.5, we explain how we shrink these zones in order to create space between adjacent zones.
4.2.1 Voronoi diagram

We chose to use a Voronoi diagram for points. This is simple to work with, but there is one drawback. If there is a long edge between multiple nodes in the same zone, part of this edge may end up in a Voronoi cell of another zone. This may bisect the zone if we are unlucky. A solution to this would be to use a segment Voronoi diagram that also supports Voronoi cells for line segments. However, an implementation for this was not readily available in Java. We decided to use the JTS Topology Suite\[20\] in order to compute Voronoi diagrams.

4.2.2 Merging Voronoi cells

After we computed a Voronoi diagram for the route graph, we want each cell in our representation to represent a zone. Since zones will likely consist of multiple nodes, and thereby multiple Voronoi cells, we have to merge the cells that belong to nodes in the same zone. This is done by taking the combined area of all cells of a zone as the area of the combined cell. This can be computed by computing the union of all individual areas.

4.2.3 Trimming zones

After we have computed the Voronoi diagram of all nodes and merged the Voronoi cells that correspond to nodes in the same zone, we want to shrink each of the merged cells when needed, so each zone will only be drawn around the nodes contained in it. We shrink each merged cell using several steps: First, we compute the convex hull of all points that are in the zone of this cell. Then we increase the size of the convex hull to not make the final zone too tight.

We do this by using an image processing technique: dilation. More about dilation can be found in books about image processing like \[21\]. Dilation can also be described as taking the Minkowski sum of a shape (in our case the zone) and a disk. JTS Topology Suite is also able to compute this Minkowski sum, so we use it for this as well. An example demonstrating dilation is given in Figure 4.3.

![Figure 4.3: An example of how dilation is applied to increase the size of a shape. The original shape is represented using a dashed outline, while the new shape is represented by the solid outline. The amount that is dilated is equal to $d$.](image)

At this point we take the merged Voronoi cell and the dilated convex hull, and compute the union of both shapes, we will call the result of this union the trimmed cell. A visual representation of this approach is given in Figure 4.4.
CHAPTER 4. ZONE VISUALIZATION

(a) First the merged Voronoi cells are computed.

(b) The convex hull is computed for each cell and dilated.

(c) The intersection between the computed convex hull and the merged cell is computed.

(d) The zones are drawn based on the intersection (details are in Section 4.2.4).

Figure 4.4: An example of how the trimming procedure is applied.

4.2.4 Drawing curved zones

Important to create smooth zones is making the trimmed cells continuous curves. We do this by using a method inspired by the drawing editor Ipe[6]: uniform B-splines. Additional information about B-splines can be found in [10] amongst others. The zones we draw are defined as closed uniform B-splines. Each segment $Q_i$ of this B-spline is a cubic spline with the following parametrization:

$$Q_i(t) = \frac{(1 - t)^3}{6} p_i + \frac{3t^3 - 6t^2 + 4}{6} p_{i+1} + \frac{-3t^3 + 3t^2 + 3t + 1}{6} p_{i+2} + \frac{t^3}{6} p_{i+3},$$

where $t$ varies between 0 and 1, and $p_i, p_{i+1}, p_{i+2}$ and $p_{i+3}$ are the control points of the spline. If we have a polygon $p_0, p_1, \ldots, p_n$ representing a zone, then we create a corresponding B-spline by combining the segments $Q_i$ for $i \in 0 \ldots n$. In order to obtain values for the final parts of the closed B-spline, we duplicate the points $p_0, p_1$ and $p_2$ to also represent the last three control points $p_{n+1}, p_{n+2}$ and $p_{n+3}$. In Figure 4.5 an example with the control points for each segment in such a B-spline is given.

4.2.5 Creating space between zones

When we draw the B-splines as computed in section 4.2.4, we only get very little space between different zones. In order to create more space, we use an image processing technique similar to dilation: erosion. More about erosion can, like dilation, be found in books about image processing like [21]. An example demonstrating erosion is given in Figure 4.6.

Now we must be careful with how much we erode from our zones; if we erode too much, one of the nodes that is part of a zone might end up outside of this zone. In order to prevent this, we
first calculate the distance $d$ between the B-spline of a zone $z$ and the node in $z$ that is closest to the B-spline. Then we erode an amount that is proportional to $d$. We chose to erode an amount equal to 0.5$d$. This ensures that none of the nodes in $z$ fall outside of the eroded shape. There is another issue with using erosion: if the zone is thin in the middle, using erosion might split the zone in two. We think that this does not occur too often, especially for a good set of zones. The main reason for this is that individual Voronoi cells in the Voronoi diagram are convex. When merging these cells, it is however possible that the zone becomes thinner. However, we believe that zones will not be thinner than the distance between the outline and the node that is closest to this outline.

4.3 Visualization using an Energy-based approach

An alternative visualization method is based on potential energy fields. An example of potential energy fields is sub-atomic interaction between protons. In order to keep the example simple, we will only consider electromagnetic forces between particles. A proton has more potential energy if it is closer to one or more other protons, because due to the repulsive electromagnetic force between protons, it would take energy to reach such a position from a position that is further away. Blinn[4] visualized surfaces in these potential energy fields where the potential energy was identical.
CHAPTER 4. ZONE VISUALIZATION

at each point on the surface; these surfaces are equipotential surfaces. A picture of equipotential surfaces in the example is given in Figure 4.7. The idea of the Energy-based approach we are proposing is to compute a potential energy field for each zone. For each field the energy field is high in regions close to graph elements of this zone, and low when far away. Then a equipotential surface of this energy field is used as the boundary between areas with a higher potential energy and a lower potential energy. The area with the higher potential energy is filled in, creating a smooth shape. In order to draw these areas, we use the BubbleSets visualization algorithm from Collins et al.[8]. The algorithm is explained in more depth in Section 4.3.1.

Figure 4.7: An example of a potential energy field surrounding two protons. Two equipotential lines are drawn around the protons. Points in the dark grey area have a higher potential energy than points in the lighter grey area. Points in the lighter grey area have a higher potential energy than points outside this area.

4.3.1 The Bubble Sets algorithm

Computing each of the zones is done in several steps and independent of the computation of other zones. We will describe the process of visualizing one zone $Z$. The first step in visualizing the zone $Z$ is to create a grid on this plane. The idea is now that we compute the potential energy for each point on the grid for each zone $Z$. This is done by summing up the energy contribution for $Z$ of all graph elements (nodes and edges) for each grid coordinate $g$. Now there are some details to this, which we will describe below. The first of these details is that we only compute the energy contribution of elements that are within a distance $d_{\text{max}}$ of $g$. This is mainly a consideration made to lower the computation time, as elements that are further away contribute much less. Now we determine the energy contribution of a graph element to $g$ using a kernel function. This kernel function computes the energy contribution $f$ based on the distance $d$ between a graph element $s$ and $g$. For determining the distance to edge graph elements, we use the shortest distance between any part of the edge and $g$. As kernel function, we use the function $f(d) = i \times (d - d_{\text{max}})^2$. Here $i$ is an influence factor variable that depends on the graph element involved. The values we use for $i$ are discussed below. Important about this kernel function is that $f(d)$ has its minimum value of 0 at $d = d_{\text{max}}$. This means there is a smooth transition between when we use the values of $f(x)$ and when the graph element is too far away and we do not add any contribution. The energy contribution given by the kernel function for varying distances is given in Figure 4.8.

This way the net energy can be calculated for every point on the grid. Now we will draw our zone $Z$ around the region of the grid that has energy $f > a$, where $a$ is the energy threshold parameter. We find the outline of this area by using marching squares on the grid.
CHAPTER 4. ZONE VISUALIZATION

\[ f = (d-d_{\text{max}})^2 \]

\[ d = d_{\text{max}} \]

\[ f = 0 \]

Figure 4.8: The solid line displays the value of the kernel function for varying distance \( d \). The dashed line indicates the value of the function \( i \times (d - d_{\text{max}})^2 \).

Values for \( i \)

Now there is one thing we have ignored so far. We introduced the variable \( i \) that is used as a scaling factor for the kernel function. The variable \( i \) is used as a weighting factor to vary the influence of different types of graph elements. The value for \( i \) depends on whether the graph element is an edge or vertex; in addition, \( i \) is a positive number if the graph element is part of \( Z \) (here an edge is part of \( Z \) if and only if both endpoints are part of \( Z \)) and negative if the graph element is not part of \( Z \). If the graph element is a node that is part of \( Z \), we use 25 as value for \( i \); for an edge that is part of \( Z \) we use 20 as value. If the graph element is not part of \( Z \) and a node, we use a value of -25 for \( i \). If the graph element is an edge that is not part of \( Z \), we use a value of 0 for \( i \) (i.e. this edge has no contribution). These values are obtained experimentally. Results of these experiments are described in Section 5.2.2. We chose to compute no contribution for edges that are not part of \( Z \) in order to deal with edges that come particularly close to parts of zone \( Z \). The biggest problem here are edges that have a node in \( Z \) as one of its endpoints. The problem with these edges is that their negative influence often creates a more concave or flattened zone outline.
Chapter 5

Results

We have tested both the Voronoi-based approach and the Energy-based approach on the network of Vienna. We chose to use the network of Vienna because of its smaller size. However, the structure of the network is still sufficiently complex. While we have also done tests on the networks of Sydney, runs with the same parameters could take between 30 minutes and 24 hours. Because of this variance, we decided to limit our tests to the network of Vienna.

5.1 Solving time

We have performed several tests to find good parameters for smoothing the solving process described in Section 3.6. In particular, we varied the termination criterion for long solving runs (As described in Section 3.7). This criterion is hard to determine because there is not clearly a best approach. We have tested the two different methods described there: Using a fixed time and using the optimality gap.

In order to test which of these criteria is more suited, we tested both with varying parameters on the network of Vienna. For testing time as termination criterion, we varied the time that was allowed before the search is terminated. For testing the optimality gap as termination criterion, we varied the threshold value for the optimality gap before termination. The results of these tests are given in Table 5.1.

In the ‘Termination’ column, the termination type is noted. For Time-based termination, this column also contains how long we tested. For example, the running time $t_{long}$ that was used for the test is equal to 10 times the running time $t_{short}$ in the row ‘Time (10x)’. For optimality gap-based termination, the number between brackets lists the optimality gap percentage. For example, the test in row ‘Gap (35)’ terminates when the optimality gap reached 35%. The column ‘Time’ lists the total running time of the solver in ‘ticks’, a deterministic measure used by Cplex. The ‘optimality gap’ column lists the optimality gap at the time of termination. The ‘Objective’ column lists the value of the objective function for the best found solution. Finally, the ‘Lower bound’ column lists the best lower bound on the objective function that is found.

First, it is pretty clear that as the time spent increases, the optimality gap decreases. This is expected, as there is more time to find a better solution or a tighter lower bound. In addition, we found that for any of the longer runs, the objective function only improves slightly, if at all. The main contribution in longer runs that lowers the optimality gap comes from tightening the lower bound. For this reason, we believe that shorter runs are sufficiently effective in finding a good solution. Because there is the possibility that a solving run will be repeated due to conflicts (as described in Section 3.6), we believe that the best termination criterion is to terminate after the solver has been running for 30 times $t_{short}$ (like measured in the row ‘Time (30x)’).
CHAPTER 5. RESULTS

<table>
<thead>
<tr>
<th>Termination</th>
<th>Time</th>
<th>Optimality Gap</th>
<th>Objective</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (10x)</td>
<td>157988</td>
<td>81.12%</td>
<td>1905.32</td>
<td>358.01</td>
</tr>
<tr>
<td>Time (30x)</td>
<td>216243</td>
<td>80.10%</td>
<td>1638.12</td>
<td>325.93</td>
</tr>
<tr>
<td>Time (100x)</td>
<td>664038</td>
<td>44.54%</td>
<td>1606.12</td>
<td>890.82</td>
</tr>
<tr>
<td>Time (300x)</td>
<td>1945435</td>
<td>34.56%</td>
<td>1606.12</td>
<td>1051.02</td>
</tr>
<tr>
<td>Gap (35)</td>
<td>2.215E7</td>
<td>35.00%</td>
<td>1680.32</td>
<td>1092.21</td>
</tr>
<tr>
<td>Gap (40)</td>
<td>1777574</td>
<td>40.00%</td>
<td>1625.11</td>
<td>975.11</td>
</tr>
<tr>
<td>Gap (45)</td>
<td>1195389</td>
<td>44.28%</td>
<td>1638.72</td>
<td>912.96</td>
</tr>
<tr>
<td>Gap (50)</td>
<td>1163815</td>
<td>49.98%</td>
<td>1636.32</td>
<td>818.41</td>
</tr>
</tbody>
</table>

Table 5.1: Results for both Time-based termination and optimality gap based termination. For time-based termination, the long running time is given by the number in brackets. For gap-based termination, the number in brackets is the gap percentage required for termination.

5.2 Visual results

In this section we compare the results from the two visualization approaches we have investigated. We analyze the results of the Voronoi-based approach in Section 5.2.1 and the results of the Energy-based approach in Section 5.2.2. For this comparison, we use the criteria defined in Section 4.1.

5.2.1 Voronoi results

Looking at the zones drawn by the Voronoi-based visualisation in Figure 5.1 shows several problems: The first problem is that often the zones are more stretched in directions where there are no other zones. This causes things like the rectangular-shaped zones. All in all, the zones are not as smooth as they could be.

In addition to this, there is a second problem for the Voronoi-based visualization. The Voronoi diagram that is generated only accounts for nodes of the graph, and does not account for any edges. This means it is possible that two nodes are part of a certain zone, but the edge between both nodes ends up falling outside of the zone. We see in Figure 5.1 that this indeed occurs in the top-central zone. This problem should disappear when the Voronoi diagram uses both edges and nodes.

The biggest advantage of using the Voronoi-based approach is that it virtually guarantees that zones will not overlap. This is because the zones are each defined by a set of Voronoi cells. None of these Voronoi cells are shared between two zones.
CHAPTER 5. RESULTS

5.2.2 BubbleSet results

Figure 5.2 shows us several things. First of all, Zones created by BubbleSets can overlap. This occurs to some extent in the top-left of Figure 5.2. When a zone is close to another zone, it maintains its circular form better than in the Voronoi-based approach. However, quite often the zones surround the nodes in them too tightly. For example, in Figure 5.3, the nodes of the center zone are arranged in an L-shape. The zone that surrounds these four nodes is also L-shaped, while an oval could also be used to cover all nodes.

In order to find the optimal set of parameters for BubbleSets, we have tested multiple sets of parameters. We have included figures showing different values for each parameter. The parameters used in BubbleSets are NodeInfluence (NI), EdgeInfluence (EI), NegativeNodeInfluence (NNI), NegativeEdgeInfluence (NEI), kernel size ($d_{\text{max}}$) and energy threshold ($\alpha$). NI and EI are weighting factors that determine how much nodes and edges respectively from a zone $Z$ contribute to the energy field of $Z$. Similarly, NNI and NEI are weighting factors that determine how nodes and edges that are not part of $Z$ reduce energy in the energy field of $Z$. $d_{\text{max}}$ determines the scale.
of the kernel used by BubbleSets. This means that if $d_{\text{max}}$ is increased, the energy contribution in the new kernel increases everywhere, instead of only in the area that was added to the kernel. The value for $d_{\text{max}}$ highly depends on the scale of the data, therefore, we cannot give a single value to do well with more than one dataset. Finally, the energy threshold $a$ determines the net energy that must be in an area to be considered part of zone $Z$. Because of the number of parameters, testing out all different combinations was too intensive.

For the variables of $NI$, $EI$ and $NNI$, we settled on the values 25, 20 and -25 respectively. For the most part, these values did not have a huge impact on the resulting image. In Figures 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 and 5.10, we show some different options for each of these parameters. We decided to set the $NEI$ to 0, as the negative influence from edges with one endpoint in the zone results in a concave zone boundary around it. An example with a nonzero $NEI$ is shown in Figure 5.11.

We decided on the value of 10 for the energy threshold $a$. Examples with a lower and higher threshold are shown in Figures 5.12 and 5.13. In addition, examples of how the zones change if the kernel size is changed are shown in Figure 5.14 and 5.15.
Figure 5.2: The Energy-based approach has been applied to a schematic map of Vienna. The zone size parameter that was used was 300 seconds. The marks on edges indicate when an edge crosses from one zone to another.
CHAPTER 5. RESULTS

Figure 5.3: A close-up of a zone of Figure 5.2.

Enhancing Schematic Maps with Zones for Improved Route Planning
Figure 5.4: BubbleSets with a NodeInfluence of 13. Reducing the $NI$ creates a bit more space between the different zones. For the rest the differences are minimal. Reducing the NodeInfluence would be a decent improvement.
Figure 5.5: BubbleSets with a NodeInfluence of 50. In this picture we can see that if we increase the $NI$, more overlap starts to occur between different zones.
Figure 5.6: BubbleSets with an EdgeInfluence of 10. The zone in the middle shows us that if we reduce the EI, zones can get very skinny around long edges.
Figure 5.7: BubbleSets with an EdgeInfluence of 30. In this picture we can see that if we increase the $EI$, more overlap starts to occur between different zones similar to increasing the $NI$. 
Figure 5.8: BubbleSets with a NegativeNodeInfluence of -13. In this picture we can see that if we decrease the NNI, overlap becomes more common.
CHAPTER 5. RESULTS

Figure 5.9: BubbleSets with a NegativeNodeInfluence of -50.

Figure 5.9: BubbleSets with a NegativeNodeInfluence of -50.
Figure 5.10: BubbleSets with a NodeInfluence of 13, an EdgeInfluence of 10 and a NegativeNodeInfluence of -13. The result is almost identical to Figure 5.2
Figure 5.11: BubbleSets with a NegativeEdgeInfluence of -20. The zone in the bottom-right shows how the negative contribution from edges can create concave zones.
Figure 5.12: BubbleSets with an energy threshold of 5. We can see zones grow and more overlap occurring.
CHAPTER 5. RESULTS

Figure 5.13: BubbleSets with an energy threshold of 15. Zones tend to shrink a bit, especially around long edges.
Figure 5.14: BubbleSets with a halved kernel size. The zones have become incredibly skinny.
Figure 5.15: BubbleSets with a doubled kernel size. There is a lot of overlap between zones and very little space between zones.
5.3 Finding a good zone size

In order to compare the results from tests with different zone sizes, we will use the last two criteria from Section 3.4. These criteria were that the zones should neither be too small and hard to count, nor too large and not distinguish between different routes. Now we did tests with varying zone sizes on Vienna. We varied the zone size between 60 and 300 seconds. The results of these tests are given in Table 5.2.

<table>
<thead>
<tr>
<th>Zone size</th>
<th>Time</th>
<th>Optimality Gap</th>
<th>Objective</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>3051</td>
<td>0.0%</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>120</td>
<td>219104</td>
<td>66.66%</td>
<td>1727.75</td>
<td>576.03</td>
</tr>
<tr>
<td>120</td>
<td>417282</td>
<td>82.15%</td>
<td>1886.75</td>
<td>336.80</td>
</tr>
<tr>
<td>180</td>
<td>232705</td>
<td>70.24%</td>
<td>1818.22</td>
<td>541.63</td>
</tr>
<tr>
<td>180</td>
<td>312778</td>
<td>71.83%</td>
<td>1728.22</td>
<td>486.93</td>
</tr>
<tr>
<td>240</td>
<td>203305</td>
<td>73.39%</td>
<td>1619.94</td>
<td>431.13</td>
</tr>
<tr>
<td>240</td>
<td>203544</td>
<td>70.29%</td>
<td>1681.44</td>
<td>488.09</td>
</tr>
<tr>
<td>300</td>
<td>216243</td>
<td>80.10%</td>
<td>1638.12</td>
<td>325.93</td>
</tr>
</tbody>
</table>

Table 5.2: Results for testing varying zone sizes between 60 and 300 seconds on Vienna. For all tests, we stopped long solving runs when the solving time exceeded 30 times $t_{short}$.

What stands out here is the solution that is found for a zone size of 60 seconds. With this zone size, the solver could find a ‘perfect’ solution; i.e. a solution where the number of zones on each route exactly represents the length of that route. Such a perfect solution with an error of 0 is actually possible in this case because in the dataset of Vienna, the travel time of each route is divisible by 60. This means you can actually locally compute the exact number of zones that any certain edge should cross by means of a simple division. Because the number of zones on each edge can be computed exactly, it is no longer needed to approximate the travel time of any routes.

We suspected that as the zone size decreases, a more fitting zone distribution can be found, resulting in a better objective value. We suspected this because the length of individual routes can be better approximated with a discrete number of smaller zones. However, our results do not suggest this is true, and even hint at the opposite. We find that as the zone size decreases, the objective value actually seems to increase. It might be that only having few zones makes finding a good distribution significantly less complex, resulting in a better solution using the time that is given.

The lower bound seems to increase as the zone size decreases. We can think of multiple causes for this. The first cause is that the optimal solution is worse for a smaller zone size. The second cause is that the problem is more complex to solve for a larger zone size. This means the solver needed more time to tighten the lower bound.

In Figures 5.16, 5.17, 5.18, 5.19 and 5.20 the visual results are shown for zone sizes of 60 seconds to 300 seconds. Each of the visualizations has been done using the BubbleSet approach.

Of these pictures we believe that Figure 5.16, with a zone size of 60 seconds, has too many zones. This makes it harder to count the number of zones on paths any user wants to compare. In the end, despite the fact that the solution found for this zone size is ideal, the resulting map is not really suitable.

On the other hand, we believe that Figure 5.19 has too few zones. The main reason is that there are cycles in the network that are contained in a single zone. This is particularly confusing because whenever a user would plan a route that travels through this cycle, it would not be clear which of the two possible paths through the cycle would be faster. An example of such a situation is given in Figure 5.21. In this case the zone in the middle does not tell if it is better to traverse...
the cycle clockwise or counter-clockwise in order to travel from A to B. Based on this, each cycle should be divided into at least two zones.

It would be even better if each cycle would be divided amongst an odd number of zones. This is because if an odd number of zones are crossed in this cycle, both paths between any two nodes on the cycle cannot cross the same number of zones. Otherwise the number of zones crossed in the cycle would be even. Because both paths between the two nodes cross a different number of zones, one of the two paths always crosses less zones than the other path. Now because criterion 2 of Section 3.4 is enforced, this route is always faster. This means that when choosing between two routes that form such an odd cycle, the zones will always show which route to pick. An example with 3 zone crossings on a cycle is given in Figure 5.22.

This leaves Figure 5.17 and Figure 5.18 as examples of a good zone size. Of these two, we prefer the larger zone size of the two, a zone size of 180 seconds. However, such a zone size would only be suitable for this graph. If travel times increase, decrease, or the map is structured differently, this zone size is likely not suitable.
Figure 5.16: The optimal solution for the schematic map of Vienna with a zone size of 60 seconds.
Figure 5.17: The schematic map of Vienna visualized with BubbleSets. One of the solutions for a zone size of 120 seconds.
Figure 5.18: The schematic map of Vienna visualized with BubbleSets. One of the solutions for a zone size of 180 seconds.
Figure 5.19: The schematic map of Vienna visualized with BubbleSets. One of the solutions for a zone size of 240 seconds.
Figure 5.20: The schematic map of Vienna visualized with BubbleSets. One of the solutions for a zone size of 300 seconds.

Figure 5.21: The zone in the middle does not tell if it is better to traverse the cycle in the middle clockwise or counter-clockwise in order to travel from A to B.
Figure 5.22: An example of a cycle with 3 zone crossings on it. In this case, the two paths on this cycle between the same nodes never cross the same number of zones.
Chapter 6

Conclusion

6.1 Results

We described a method to augment schematic maps in order to help users determine which route is the shortest route between any source and destination. In order to do this, we augment the map with smooth-shaped zones that each represent a certain amount of time to travel through. In order to find a set of zones that is effective, we first determine the optimal location on the graph for every zone. Then we draw each zone using either a Voronoi-based approach or an energy-based approach.

A number of shortcomings however still remain:

First of all, most zones are not as simple in shape as they could be. For example, there are many zones that have a bit of an odd shape, while an oval covers nearly exactly the same area. Adding volume to a zone in order to make it more oval is also often in many cases possible. For example, an L-shaped zone can often be turned into an oval that covers the nodes (as shown in Figure 5.3). We have not found a way to do this automatically however.

There is also some overlap between the zones found in the Energy-based approach. For this reason, it is worth looking into modifying the BubbleSets algorithm in order to try and eliminate overlap whenever possible.

Compared to the Energy-based approach, the Voronoi approach produces less curvy zones. However, there is no overlap possible between multiple zones.

In addition, the original sketches made by Haverkort had zones covering the entire map. This includes zones that did not have any nodes inside them. These empty zones can help to show the zone density in an area of the map, neither the Voronoi-based approach, nor the BubbleSet approach does generates these empty zones. However, both approaches fill empty space in the inside of the route graph pretty efficiently. Therefore, we think it is not needed to add them.

6.2 Future Work

Visualization alternatives

In this thesis we have explained two methods of visualizing zones on route graphs. However, many alternatives exist. Vehlow et al.[26] have done an extensive survey on visualizations of group structures on graphs. In the context of this survey, our problem is to visualize disjoint groups where the groups are flat (i.e. there is no hierarchy). Section 5.3.2. of the paper describes the visual
result we are trying to accomplish. It is worth looking into visualization methods described there.

A more specific survey is the survey written by Galton et al.\cite{galton2012}. They have analysed different methods to represent a set of points in two dimensions by the region these points occupy. Of particular interest is the second half of the paper, where Galton analyses the ‘area of influence’ of a set of points. In here, two different Voronoi-based approaches are analysed alongside several other approaches. One of these Voronoi-based approaches is an approach from Alani et al.\cite{alani1998}. This approach is interesting in particular due to the similarity between it and the Voronoi-based approach described in this thesis.

Voronoi adjustments
As said in Section 4.2.1, a good way to improve the quality of the Voronoi diagram would be to use a segment Voronoi diagram. However, instead of using a segment Voronoi diagram, with all the complications involved, a simpler alternative is to replace each edge with several nodes for the purpose of computing the Voronoi diagram. Since edges are usually not too long, we think that this is a good approximation for a segment Voronoi diagram.

There is one main reason why the zones of the Voronoi-based approach are less smooth than those created by BubbleSets: The Voronoi diagram gets more complex as more different equidistant lines from the diagram meet in a certain region. When this happens, any zones that are bounded by that complex region follow the diagram more closely than needed, due to how B-splines are defined. An example of this problem is shown in Figure 6.1. A relatively simple solution might be to introduce extra points in the cell in order to equalize the distance between subsequent polygon points in the cell. This will however cause the B-spline to follow the cell contour more closely, likely resulting in a less smooth zone. On the other hand, simplifying the Voronoi diagram such that the equidistant lines mentioned above meet in exactly the same point instead of multiple points close together may be possible. However, we do not know a good threshold for when to merge several points, and when not to merge points. Even then, we are not sure if using a threshold solves the problem. It may be the case that points that are farther apart than the threshold can still cause problems, regardless of the threshold.

Figure 6.1: A red zone is drawn for the near-rectangular Voronoi region in black. Counter-intuitively, the zone is more stretched out towards the top-right corner, while the Voronoi region is smaller there.

Another solution that could improve the shape of zones might be to change the order of several steps in the visualization. In the current visualization, we compute the curve-shaped B-spline
representation based on the trimmed cell before eroding this B-spline to create space between the
different zones. However, it is also possible to shrink the trimmed cell first with erosion and base
the B-spline representation on the eroded cells. The advantage of doing this is that the resulting
zone maintains the shape of the trimmed cell better. However, a disadvantage is that the B-spline
is computed after eroding. Since the eroded shape is smaller, there is a bigger chance that a node
of the route graph may end up outside of the B-spline, and thus outside of the zone. We think it
is worth investigating whether or not a node ends up outside of its zone often enough because of
this. If that does not occur often, the alternative order could be better.

**BubbleSet adjustments**

There are still some things we can change to hopefully improve the Energy-based BubbleSets
approach. First of all, we can change the kernel function BubbleSets uses. Currently we use a
quadratic kernel function, but possibly other kernel functions might be more effective. In addition,
mixing different kernel functions for edges and nodes could be effective.

Second, when we compute the distance between the point \( p \) we are calculating energy for and
an edge, we use the minimum distance between \( p \) and this edge. Instead of the minimum distance,
we have thought about integrating the distance over the edge. This way, if only a small portion of
the edge is close, the contribution will not be as large. The integral can be computed algebraically
for some kernel functions. However, the distance to different points on the line depends on the
orientation of the edge in relation to \( p \).

There is also the problem of zones from BubbleSets possibly overlapping. One solution to this
might be to extend the BubbleSets algorithm by adding negative contributions for other zones.
To be specific, the affinity for a specific area for a specific zone is decreased when another zone is
close to this area. This does create complications, because the area other zones cover may not be
calculated yet. It might still be possible to use information about the area of other zones by using
an iterative approach; in one iteration of BubbleSets, the zones are computed by using data from
zones computed in the last iteration. After multiple iterations, the zone locations will hopefully
stabilize in order to find a final zone distribution.

**Odd Cycles**

When looking at finding a suitable zone size for the zones, we found a very useful relation between
the cycles in the graph and the effectiveness of the zones for choosing between routes. If traversing
a certain cycle requires an odd number of zone crossings, then the two paths that follow the cycle
between two nodes on the cycle cannot cross the same number of zones, and thus not have the
same zone length. Because of this and because criterion 2 in Section 3.4 is enforced (which states
that if a path has a shorter zone length, it must be shorter), the path that has a shorter zone
length is never slower than the other path. As a result, if there are more odd cycles in the graph,
then the map becomes more effective at showing the shortest route between nodes. We believe it
is worth investigating how the cycles affects the number of different paths between destinations
that cross the same number of zones.

**Solving Improvements**

Currently the MIP is very large and takes a long time to solve. Because of this, the time required
to get a good solution for larger maps is long. In order to reduce the time required, we have several
suggestions for improvement.
First, it is possible to 'prune' the travel network by removing all nodes that only have 2 edges. This essentially means these two (or more) edges are replaced by a single edge. This way, the simplified graph can hopefully be solved much faster. When a solution for this simplified graph is found, the graph can be expanded again by re-introducing these nodes. When the nodes are re-introduced, we can compute to which zone on the route this node belongs. An example of this process is shown in Figure 6.2. It could be worth testing if simplifying the graph like this allows for faster solving, or if simplifying the graph does not significantly reduce the complexity.

Figure 6.2: A sketch of how to simplify the graph and solve the simplified problem. First the graph is simplified. Then the zone locations are determined for the simplified graph. Finally, the omitted nodes are reintroduced. At this point, the computed zones are distributed along the route.
Bibliography


[26] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. The state of the art in visualizing group structures in graphs. EuroVis. 57