A face-based approach for automatic metro-map generation

Zhang, Lirui

Award date:
2021

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Algorithms, Geometry, and Applications Research Group

A face-based approach for automatic metro-map generation.

Master Thesis

Lirui Zhang

Supervisors:
Arthur van Goethem

Committee Members:
Arthur van Goethem
Wouter Meulemans
Erik de Vink

Eindhoven, October 2021

Abstract

We explore three different face-based approaches to automatically generate metro maps. The first two approaches give unsatisfactory results, but they help to discover the underlying problems. The results in final approach has significant opportunities to improve the readability of metro maps.

The first approach is called Grid Map approach. We embed the metro map into a grid map to help ensure an octilinear layout. We want to increase the roundness of the face to leave more space for labels and to separate the close routes. To do this, we calculate the centroid of a face and draw a circle with the centroid as the center. We move the vertices inside the circle away from the centroid and move the vertices outside the circle toward the centroid. Every vertex has five candidate positions and they can move one grid each step. We hope to move them onto the circle so that the face will have the largest roundness. However, we have to make sure the topology is unchanged every time we move a vertex and all the incident edges of the vertex have to be octilinear. This put too much restrictions on the location selection for the vertices and consequently there map be sharp bends in the final result. Besides, for some vertices, none of the five candidate positions can be used and these vertices can not be optimized at all. Thus, we decide to change our research direction.

The second approach is called Attach Circle approach. We abandon the grid map and the octilinear layout in this approach. We drag the vertices on a face directly onto the circle. We first divide the circle evenly, and then find a target point for each vertex. We move the vertices one by one by a small step to their target points. If after one step a vertex breaks the topology, we move it back this small step. This approach can increase the roundness of the faces and reduce the clusters. However, when two adjacent faces are too far away, their common vertices have to attach to the two far-away circles at the same time. Eventually the common vertices will fall near the middle of the two circles, far from their target points in both circles. This causes long edges, making the edge length of the metro map more unbalanced. Thus, we have to propose another approach to deal with the long edges.

The final approach is called Force-Directed approach. The long edges are caused by the poor distribution of centroids, so we first rearrange the centroids to obtain a better centroid distribution. We use a force-directed method to make adjacent *FaceCircles* to be tangent and rotate the centroids. Then we use the collapse-and-reinsert method to first place the degree-3+ vertices and then reinsert the degree-2 vertices on the chains between the degree-3+ vertices. We process the outside vertices and the inside vertices in different ways. The inside degree-3+ vertices are placed at the junction of their adjacent faces, and the distances from the adjacent centroids are proportional to the number of vertices on the faces. The outside degree-3+ vertices are placed on the apollonian circle of their adjacent centroids. The inside degree-2 vertices are reinserted on the apollonian circle arc of their adjacent centroids and two arcs that can connect the apollonian circle arc with the two degree-3+ vertices smoothly. The outside degree-2 vertices are processed in a similar way to the inside degree-2 vertices, but replace the apollonian circle with the *FaceCircle*. We add a box around the metro map to include the dangling edges in a face to get a real metro map. We tested this approach with five metro maps. In each metro map, the variance of the edge lengths is reduced, meaning that the distances between the stations are more even. The roundness of the faces increased, and the faces obtain a more regular area distribution. We believe our final approach has significant opportunities to automatically generate the metro maps with higher readability and usability.

Preface

My deepest gratitude is first to Arthur van Goethem, my supervisor, for supervising me studying this interesting topic of face-based approach for metro-map generation. He is one of the most responsible and lovely teachers I have ever met. He always gives me guidance and confidence when I am confused and get stuck. Brainstorming new algorithms with him is the most interesting thing, and he has given me a lot of valuable suggestions for the algorithms. He also uses his personal time to revise my thesis and gives me very useful feedback. Without his constant and enlightening guidance, this thesis would never have been done.

Secondly, I would like to thank my friend HaiXusun, who gives me many suggestions on the algorithm implementation. He helps me to solve some problems in the code through debugging, which saves me a lot of time.

Finally, I would like to thank my dear parents for their love and financial support. They help me a lot in my life and give me a lot of encouragement. This allows me to concentrate on my thesis without distractions.

Contents

Chapter 1

Introduction

In this chapter, we first describe the problem of metro map creation and make a formal definition of our research in Section 1.1. Then we give a brief history of metro map design in Section 1.2, which was the first stage in the creation of metro maps. Next we introduce the evolution of metro map optimization in Section 1.3, which is the second stage of creating metro maps. In Section 1.4 we talk about our innovations and major contributions and in Section 1.5 we provide an overview of our thesis.

1.1 Problem description

Metro is an important means of travel for passengers and is essential to the public transportation system in big cities. After nearly two centuries of development, the metro network is not just a means of transport in itself, it is also an important symbol of the city's uniqueness. When looking at the metro network schematic diagram shown in Figure 1.1, we can easily identify which city each metro network corresponds to.

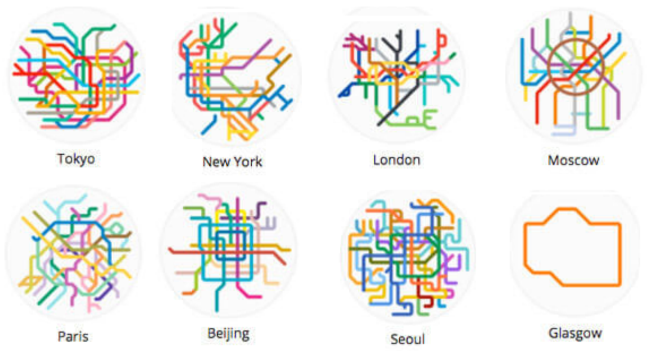
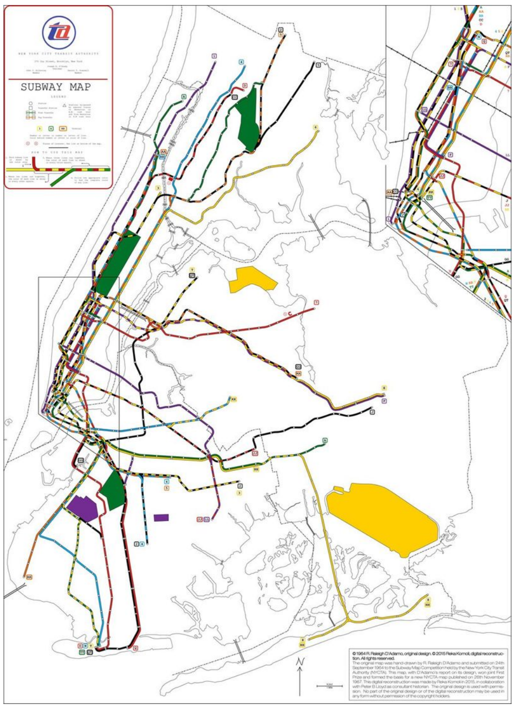
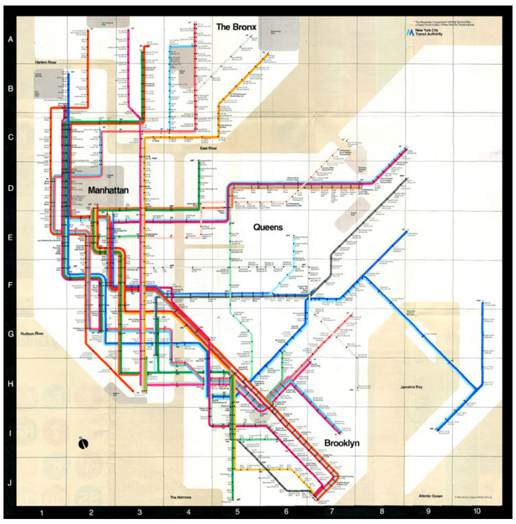


Figure 1.1: An example to show metro networks in different cities. If people see a circle in the middle of a metro map, and the other routes cross the circle diagonal, it is easy to know that this is a metro map of Moscow. Similarly, when there are two nested squares in the center of a metro map and other routes run horizontally or vertically through the square, one can easily recognize it is a metro map of Beijing. Figure form [9].

In order to show the metro network in more detail and facilitate people's navigation, the metro maps were created. However, designing a metro map is one of the most difficult graphic design tasks because there are hundreds of stations distributed unevenly and dozens of routes cross each other. In urban centres, stations tend to be particularly close to each other, with dense routes overlapping each other, while in the suburbs the stations are sparsely distributed and have only one or two routes. The design of metro maps is very important for passengers' navigation. Bad metro map design, as is shown in the New York metro map in Figure 1.2a, routes are very close and overlap each other, making it difficult to track routes and distinguish stations. Such a poorly readable map often confuses passengers, for it does not even show the names of the stations and its clustered routes are easy to misread. On the contrary, good subway design not only meets the design aesthetic criteria, but also has the practicality to facilitate passengers navigating, as shown in Figure 1.2b of another New York metro map. Passengers can easily track each route, distinguish different stations, quickly locate the destination and expediently choose their interchange stations. Therefore, designing an aesthetic and practical metro map is a very challenging but important task.



(a)



(b)

Figure 1.2: An example of two different metro map designs for New York to show why metro map design is important for navigation. (a): A bad designed map of the New York metro map. On the left side of the map, dozens of routes overlap and wrap together, making it difficult to distinguish stations and track routes. The name of the stations are not marked on the map because the crowded route and clustered stations leaves no room for labels. Passengers are often confused when using such maps to navigate, and it is easy to misread the routes and get lost. Figure from [8]. (b): A good designed map of the New York metro map. The map uses octilinear layout, with routes horizontal, vertical, or diagonal, and each route is represented in a different color, making it easy to track. In addition, the name of each station is marked next to it, and the distance between stations is balanced, making it easy for passengers to find their destination and make transfer plans. Figure from [1].

Creating a metro map usually involves three stages, respectively the stages are design, optimization and beautification. The design stage is the first stage of creating a metro map. During the design stage, the designers set aesthetic criteria that the map should meet, such as layout type (octilinear or hexagonal), distance between stations (fixed or arbitrary length), and the shape of the routes (straight line or curve). As early as 1933, Harry Beck developed a set of aesthetic criteria for metro maps while designing London metro maps, including removing useless details, using octilinear layout, keeping smooth straight routes, distributing stations evenly [15]. We will discuss these design criteria in more detail later. Beck's design criteria profoundly improved the readability and practicability of metro maps and were imitated by other cities over the next hundred years.

The next stage is the optimization stage. During the optimization stage, taking the original geographic location as the initial input, the location of stations and routes are adjusted to get a more even feature spacing. In each iteration, the stations and routes are moved by a small distance to better meet the aesthetic criteria, while keeping the topology unchanged. Previous metro map optimization was done manually, and cartographers embedded the original metro map in the grid map, adjusting the location of each station with the grids [28]. In recent years, computer-assisted metro map optimization has begun to develop. Computer scientists use algorithms to control computers to automatically adjust the location of stations and routes while trying to satisfy the design criteria.

The last stage is the beautification stage. During the beautification stage, different routes on the metro maps are embellished with lines of different thickness and color, and different station types are marked as different symbols. For example, transfer stations are represented by a large red circle symbol in bold, while ordinary stations on the road are represented by a small black dot. In addition, this stage determines the style of labels and legends, including font style, color, size, and so on.

The formal definition of our research

We want to create practical and beautiful metro maps for different cities through the above stages and our main research at each stage is described here. During our design stage, we adapt Beck's design criteria according to our research needs. We want our metro map stations to be evenly distributed, with smooth and non-intersecting routes, which are in line with Beck's design criteria. But we do not necessarily choose octilinear layout. We want to compare the octilinear and curvilinear layout and choose the one that works better.

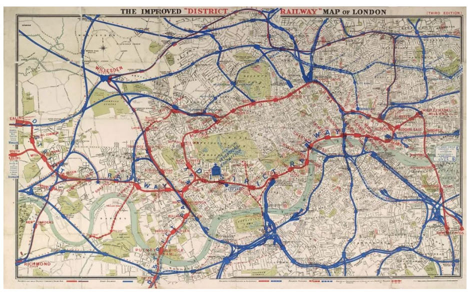
Then we mainly focus on the optimization stage. Traditional automatic optimization algorithms improve the feature spacing of metro maps by directly adjusting the location of stations, which are called 'vertex-based' algorithm. These algorithms focus on the stations and routes to see if they are too close or too far away and then move them directly. Many 'vertex-based' algorithms algorithms are inefficient for complex metro maps because the distance between every two stations has to be calculated in each iteration to determine whether to move. However, we observe that the number of faces (closed regions surrounded by routes) in the map is much smaller than the stations. Even for complex maps with hundreds of stations, there are only a few dozen faces. Therefore, we think that we can first distort the faces to have regular size and shape, and then just place the routes and stations at the border of the faces. Because the number of faces is much smaller than stations, we assume that this method would be more efficient. Besides, although the existing 'vertex-based' algorithms can more or less optimize the layout of metro maps, all of these algorithms that directly adjust hundreds of stations have disadvantages. Some algorithms do not guarantee octilinearity, while others do not obtain even distribution of stations. Thus, we want to explore the performance of the new algorithm that indirectly adjust stations through faces in dealing with these disadvantages. We hope to bring new research ideas for metro map optimization. Therefore, our main focus in this thesis is presenting an automatic optimization algorithm to distort the faces to improve the feature spacing of the metro maps. Our algorithm that focuses on faces is called 'face-based' method. And just like 'vertex-based' method, we adjust our faces iteratively. We adjust the sharp corners in the faces to make them flat, and we adjust the size of the faces to make them regular, such as making the area to be proportional to the number of stations.

As for the last beautification stage, since we only represent the metro map with geometric shapes, beautification is not the emphasis of our thesis and will not be discussed.

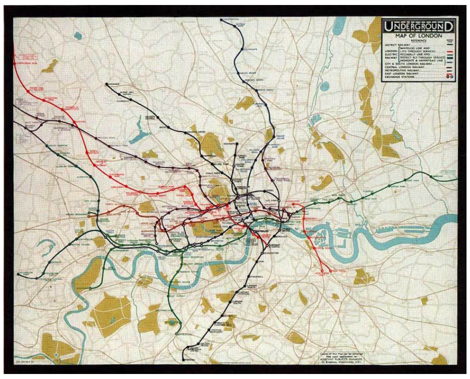
1.2 A brief history of metro map design

The design stage is the first step in creating metro maps, which is crucial for the final effect of the metro maps. The earliest metro maps are very complex that includes not only stations and routes, but also rivers, building details and parks at their real geographic location (see Figure 1.3a). Although very precise, these details in the metro map are not practical and cumbersome for passengers. People can know what parks and buildings are around each station, but they do not know the name of the stations or at which stations they can change lines.

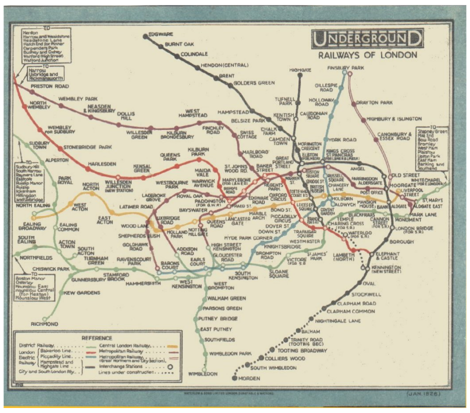
Later, designers removed some details and retained only stations, routes, and iconic geographic features, such as rivers (see Figure 1.3b). However, this map still requires that the map and the actual geographic location correspond accurately. The central area of the map is overcrowded, the boundary region wastes a lot of space and the crossed routes cause confusion and disorientation. When passengers' eyes track along one route, it is likely to mistakenly switch to another route at the intersection. So the readability and usability of this metro map is still not high.



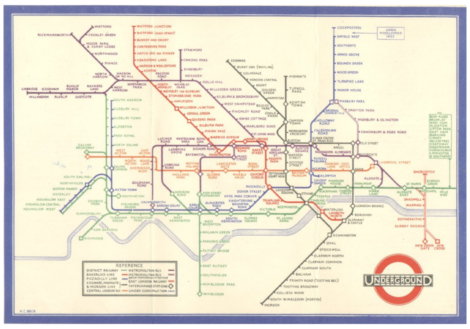
(a)



(b)



(c)



(d)

Figure 1.3: The development of metro map design. (a): The early metro map that contains not only stations and routes but also rivers, building details and parks. Figure from [7]. (b): The early metro map that requires the map and the actual geographic location should correspond accurately. The central area is clustered, but the boundary region waste a lot of space. Figure from [5]. (c): Fred Stingemore's metro map for London. The distance between stations is more even and the space between close routes is slightly larger. Figure from [6]. (d): Harry Beck's metro map for London. Beck ignored the real geographical location and used only horizontal, vertical and diagonal lines. He also made the distance between stations equal to avoid the clusters. Figure from [4].

Then in 1926, a cartographer named Fred Stingemore improved the London metro map by further simplifying the metro maps (see Figure 1.3c). He abandoned all elements that were useless to the metro map used as navigation, leaving only stations and routes. Besides, he moved the stations from their real geographic location and made the distance between the stations approximately equal. This can make the stations more evenly distributed. The routes were also slightly deviated from their actual geographic locations. Two routes that were very close to each other were moved some distance apart so that the space between them increased and they were overlap-free. Besides, he shortened the length of long routes in suburbs and enlarged the downtown area. These changes improved readability over maps that were entirely geographically based.

The breakthrough in metro map design came in 1933, when an idea by Harry Beck completely changed the conventional metro maps. Beck thought that it was not the geography of the stations that matters to passengers, but the interchanges between routes. In an interview in 1985 with Ken Garland [15], author of the book *Mr Beck's Underground Map*, Garland said that passengers in the metro were in the underground 'Tube'. So passengers are unconcerned about the real geographic elements above ground or the real distance between two stations, what is important are the connections between the 'tubes' and at which stations they should transfer. This idea is critical to Beck's design. To achieve this, Beck ignored the actual geographical location, instead he used all horizontal, vertical or diagonal lines to depict the topology of stations and routes. In addition, he specified that each route should have minimal bends and distributed the stations by equal distance along the routes (see Figure 1.3d). The center of the metro map is no longer clustered, the boundary region is used properly, and the routes are smooth and easy to track. This design greatly improved the readability of the metro map, and became the classic design of metro map in the future.

Beck's design has been widely adopted around the world for decades and it has become an universal design language for metro maps. Over the last two decades, many designers further improved and deformed on Beck's basis and have designed more metro map layout styles. Wu et al. [28] summarize some existing design styles of metro maps and Figure 1.4 is an example to show several different layout designs of the same metro map. *Curvilinear layout* means stations are connected by smooth curves. *Schematised Curvilinear layout* means stations are adjusted to the appropriate location and then connected by smooth curves. *Concentric circles layout* means each edge follows a radial mesh consisting of a set of concentric circles and lines from the center of the circle. *Multilinear layout* means all edges are straight lines but can be in any direction. *Octilinear*

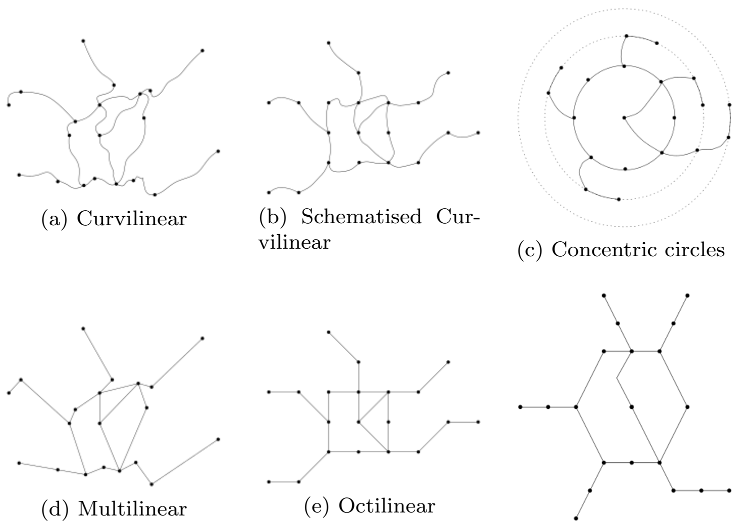

(a) Curvilinear (b) Schematised Curvilinear (c) Concentric circles
(d) Multilinear (e) Octilinear (f) Hexalinear

Figure 1.4: Several different layout designs of the same metro map.

layout means the incident edges of each vertex have to be horizontal, vertical or diagonal lines. Thus, the angle between each edge and x-axis is an integer multiple of 45 degrees. *Hexalinear layout* is similar to octilinear layout. The incident edges have to be horizontal or at an Angle of 60 degrees from the horizontal line.

In addition to these design styles, there have been styles designed specifically for research and visual effects in recent years, such as the metro maps in Figure 1.5. In all of the designs, Beck's octilinear layout is the most popular one, but each design has its own advantages and disadvantages, designers can choose according to their needs.

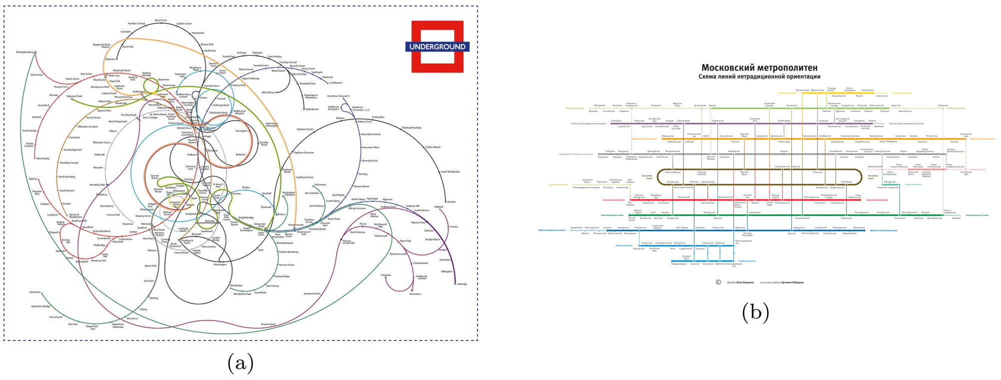


Figure 1.5: Metro map styles only used for research and visual effects. (a): A design style of London metro map by Francisco Dans. Figure from [2]. (b): A design style (Rectilinear) of Moscow metro map by Ilya Borisovich Birman. Figure from [10].

1.3 A brief history of metro map optimization

The optimization stage is the second step in creating metro maps and determines the final results of the maps. The initial metro map optimization was done manually and Wu et al. [28] describe this process in their paper (Figure 1.6). First embed the original map into a dense grid map, then iteratively adjust the location of each stations in the grid according to aesthetic criteria, and finally remove the grid map and add legends. This was an extremely time consuming process, as the metro map had to be redrawn every time a station was moved. Therefore, as metro maps became more complicated, drawing maps manually became an extremely challenging task. In addition to being boring and time-consuming, it was also difficult to determine which points on the grid a station should go to and where the routes should be bent when drawing by hand. And it was difficult to ensure that stations were evenly distributed while maintaining the topology.

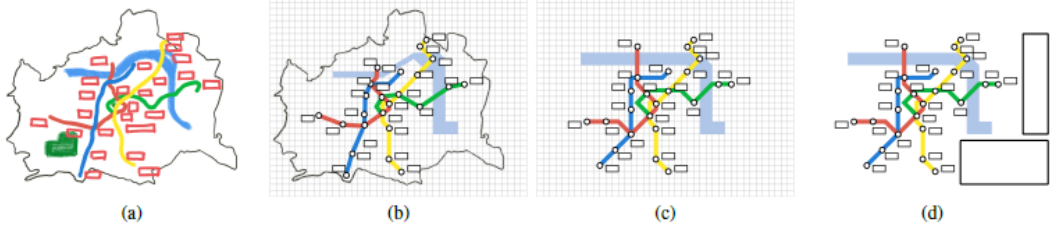


Figure 1.6: The time-consuming manual drawing process: first embed the original map into a dense grid map, then iteratively adjust the location of each stations in the grid according to aesthetic criteria, and finally remove the grid map and add legends. Figure comes from the paper of Wu et al. [28].

Therefore, automatically optimize metro maps with computers came into being.

At first, there was no algorithm specifically for metro maps optimization. Until about 2000, the algorithms that existed were all about the general schematization of geographic maps. Map schematization abstracts important information from the geographic map's original shape and neglects irrelevant details, such as Figure 1.7 shows. The algorithm can determine which points are relevant for the shape of the map and remove the unimportant points to abstract the map to a polygon. In 2000, Barkowsky et al. [12] first applied map schematization algorithm to metro maps and was believed to be the first person attempt to automatically optimize the metro maps [22]. Their results for Hamburg metro map is shown in Figure 1.8. The bends of the routes are reduced and the topology is maintained, but there are intersections and clusters in the result metro map.

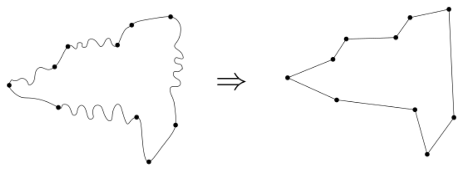


Figure 1.7: An example to show map schematization.

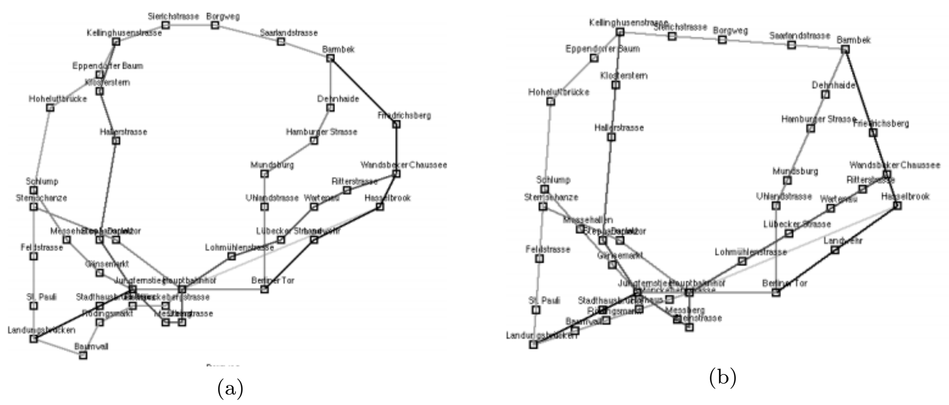
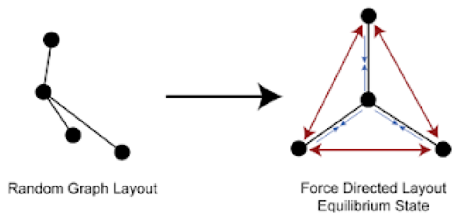


Figure 1.8: The first attempt to apply the map schematization algorithm to metro maps. (a): Original input for Hamburg metro map. (b): Result for Hamburg metro map in Barkowsky et al.' paper. Figure from [12].

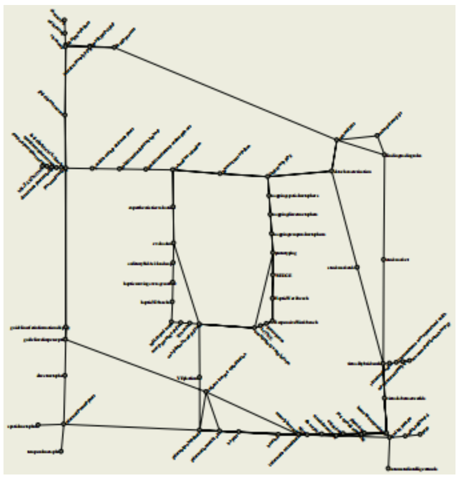
Later, algorithms that are specifically designed for metro maps started being developed by people. Hong et al.[19] proposed a new automatic algorithm for metro map layout problem. They studied various hand-drawn metro maps that existed at that time and summarized their findings into five criteria that a good map should meet. The five criteria were straight lines for edges, no edge crossings, no overlapping, octilinear layout, and each line had an unique color. Then they presented a force-directed method to automatically distort the metro maps to meet these criteria. They regarded routes as springs, stations as spring connections and gave an optimal distance between two stations. Two stations close together were pushed by a repulsive force and two stations far apart were pulled by an attractive force, and eventually the whole map system would be dynamically balanced (see Figure 1.9a). Hong et al.' algorithm can give a much better result (see Figure 1.9b) than the algorithm of Barkowsky et al. that we show before (Figure 1.8) and the interesections and clusters disappear. However, the stations are still not evenly distributed and the layout is not exactly octilinear.

Then Daniel Chivers and Peter Rodgers [16] improved Hong et al.' algorithm to obtain a well distributed and octilinear metro map. Hong et al. combined the standard spring embedder forces with an octilinear magnetic force, but these forces conflict with each other, making the metro map close to but not exactly octilinear. However, Daniel and Peter varied the forces. They emphasize the standard forces in the beginning to produce a well distributed graph, with the octilinear forces becoming prevalent at the end of the layout. The result of Daniel and Peter' algorithm is shown is Figure 1.10. The stations are evenly distributed and the edge lengths are balanced, which is much better than Hong et al. result.

Now there are many algorithms of metro map optimization, many of which can give satisfactory results. We will discuss them in detail in Chapter 3. And we will study them and adapt them into algorithms that fit our 'face-based' method.



(a)



(b)

Figure 1.9: An example to show Hong et al.' force-directed method that is specifically for metro map optimization. (a): Two close stations are pushed by repulsive force and two stations far apart are pulled by attractive force, and eventually the whole map system would be dynamically balanced. Figure from [3]. (b): Result of Hong et al.' algorithm.

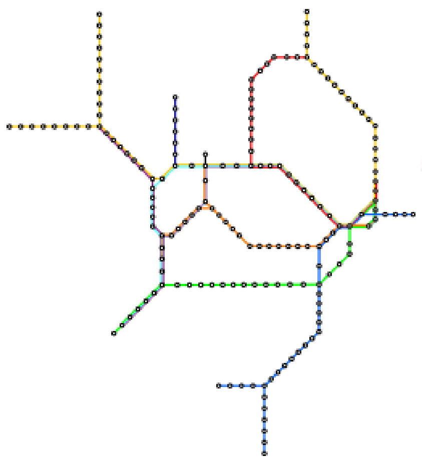


Figure 1.10: The result of Daniel Chivers and Peter Rodgers.

1.4 Our contribution

All existing automatic optimization algorithms are 'vertex-based' methods, that is, map optimization is realized through direct manipulation of vertices(stations). We are the first to optimize metro maps using 'face-based' method. We want to explore whether this method can produce beautiful and practical maps, thus bringing new ideas to map optimization. Besides, since the number of faces is much smaller than the number of vertices, we want to test if 'face-based' method can improve algorithm efficiency. In addition, we want to compare the results of 'face-based' method with 'vertex-based' method to see the advantage and disadvantages of both method.

1.5 Outline of this thesis

The structure of this thesis is as follows. Chapter 1 introduces our research topic and our contribution. Chapter 2 introduces some definitions and relevant knowledge that will be used in this thesis. Chapter 3 lists some previous work related to the automatic optimization of metro maps and discusses their strengths and weaknesses. Chapter 4 describes and discussed our first approach and why it can not work. Chapter 5 describes and discussed our second approach and why we think it unsatisfactory. Chapter 6 describes and discussed our final approach and discusses its advantages and disadvantages. Chapter 7 is our conclusion and our plan for the future work.

Chapter 2

Preliminaries

In this chapter,we first review the basic geometry and the calculations of them in Section 2.1. We review what is graph and some special terms about graphs in our thesis. We also review the construction of faces from a graph and the calculation of the Apollonian Circle. Then in Section 2.2 we introduce some relevant background knowledge about the metro maps that we use in this thesis, including the metro map design criteria and the force-directed graph drawing.

2.1 Basic Geometry

2.1.1 Graphs

A graph is an data structure to model relationships between a set of objects. An undirected graph $G = (V, E)$ consists of a set of vertices V and a set of edges E where E are the line segments between V. Two vertices u and v that are connected by an edge uv are called *adjacent* and u and v are called *neighbors* to each other. We denote the neighborhood of v to be $N(v)$, which means all vertices that adjacent to v. The degree of a vertex v is the number of its neighbors and is abbreviated by $deg(v)$. Edge uv is called an *incident edge* of u and v. The number of incident edges of a vertex is equal to $deg(v)$. We call the closed region surrounded by vertices and edges to be a *face*. The face is called the *incident face* of all vertices and edges on it. Faces that have at least one common vertex are called *neighbors*.

In our thesis, we use graph to represent the metro maps and some specific definitions of our thesis are given here and we use Figure 2.1 as an example to illustrate. *Vertices* are the stations of the metro map, for example vertex u, v and p. *Edges* are the routes of the metro map, for

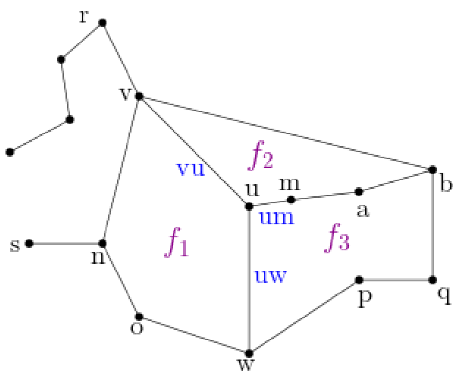


Figure 2.1: An example graph. For vertex u, vertex v, w, m are its neighbours and edge uv, uw, um are its incident edges. Face f_1, f_2, f_3 are its incident faces. The degree of u is 3. Edges uv, uw, and um are neighbors to each other. Faces f_1, f_2, and f_3 are neighbors to each other.

example edge uv, pq and vr. *Inside faces* are the faces that are surrounded by edges forming a closed loop, for example face f_1, f_2 and f_3. The *outside face* is the face that around the entire metro map. There is only one outside face f_i. An *inside degree-3+ vertex* is the vertex that has degree greater than or equal to 3 and all incident faces are inside face, for example vertex u. An *outside degree-3+ vertex* is the vertex that has degree greater than or equal to 3 and adjacent to the outside face, for example vertex v, n, m and w. *Chains* are the sequences of degree-2 vertices between two degree-3+ vertices, for example, chain $[wpqm]$ and $[vm]$. *Inside chains* are the chains that all incident faces are finite faces, for example chain $[um]$ and $[uv]$. *Outside chains* are chains that are adjacent to infinite face, for example chain $[vm]$ and $[wpqm]$. *Dangling edges* are the edges which either stand alone, or are attached only to one vertex, for example edge vr and sn.

2.1.2 Construct faces on the map

Our focus is on the 'face-based' method, so the first thing is to find all the faces in the graph. We first sort the incident edges for each degree-3+ vertex counterclockwise and obtain an incident-order-list (see Figure 2.2a). Then we start at any vertex and traverse the edges of a face clockwise. Whenever a degree-3+ vertex is encountered, always select the next edge of the edge that we have just traversed according to the incident-order-list. Then we continue to traverse until we encounter the vertex where we start, and we find a face.

It is important to note that constructing faces on metro maps in this way will loss the dangling edges, for we do not deal with the outside face. Thus, after this step, our metro map will look like the one shown in Figure 2.2b. But we will propose a method to handle dangling edges and the stations in subsequent chapters. Finally we successfully add the dangling edges back and get a complete metro map.

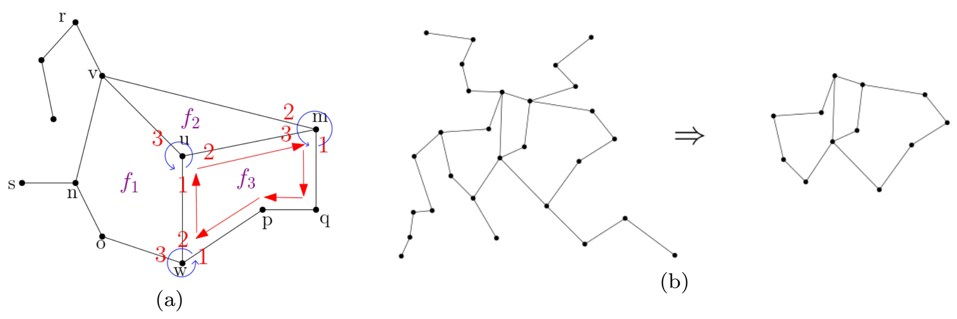


Figure 2.2: An example to show constructing faces on the metro maps. (a): First sort the incident edges for degree-3+ vertices in counterclockwise order. The incident-order-list for vertices m, u and w are all $1->2->3->1$. Then we start at vertex q for example. We traverse clockwise through p and until we meet w. The edge we have just traversed is $pw(1)$, and the next edge in the incident-order-list of w is $wu(2)$. Again at vertex u, the next traverse edge in the incident-order-list of $wu(1)$ is $um(2)$ and at vertex m we should traverse from $um(1)$ to $mq(1)$. We meet q again, and then we get face f_3. (b): Dangling edges will be lost in this step.

2.1.3 Apollonian Circle

In our algorithm, we use the Apollonius circle to show the proportional force of two faces. Thus, we will review the calculation of Apollonius circle here.

The definition of Apollonius circle is: the trajectory of a moving point on the plane whose distance ratio to two fixed points is equal to a constant k. As is shown in Figure 2.3, $A(x_1, y_1)$ and $B(x_2, y_2)$ are two fixed points, and the distance between A and $P(x, y)$ is d_1, the distance between B and $P(x, y)$ is d_2. When moving P, we require that d_1/d_2 will always be k, that is, $\frac{d_1}{d_2} = \frac{|PA|}{|PB|} = \frac{\sqrt{(x-x_1)^2 + (y-y_1)^2}}{\sqrt{(x-x_2)^2 + (y-y_2)^2}} = k$. Then the trajectory of P will be a circle named Apollonius circle, and the center of this circle is $(x_{Apollo}, y_{Apollo}) = \left(\frac{x_1 - k^2 x_2}{1-k^2}, \frac{y_1 - k^2 y_2}{1-k^2}\right)$, while the radius of this circle is $r_{Apollo} = \left|\frac{k}{1-k^2}\right| \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$.

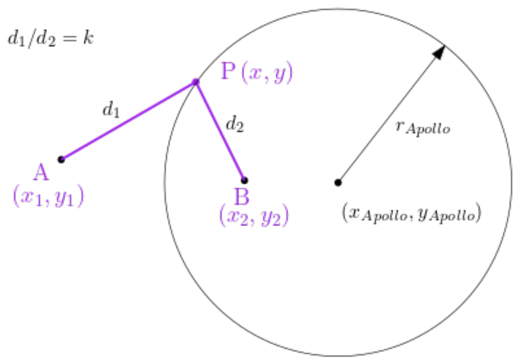


Figure 2.3: An example of Apollonius circle. A and B are two fixed points, and the distance between A and P is d_1, the distance between B and P is d_2. If when moving P, d_1/d_2 will always be k, then the trajectory of P will be a circle and we can calculate its radius and center coordinates.

2.2 Metro maps

In this section, we introduce some background about metro maps, including the common metro map design criteria, what is the protection of topology and the force-directed graph drawing.

2.2.1 Metro map design criteria

As discussed in Chapter 1, The appropriate design criteria is vital for modeling the metro map problem. When looking at Ovenden's book [24], we can find that most of the maps meet some common principles. Nöllenburg summarized these principles in his paper[22]:

C1 Metro maps should have octilinear design style, which is the most common metro map layout today.

C2 The embedding of the map must be preserved, which means keeping the topology when moving the vertices and edges.

C3 The bends in the routes should be as few and smooth as possible to facilitate eye following.

C4 Distances between adjacent stations should be as equal as possible to reduce clusters.

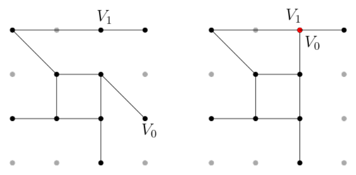
C5 Leave enough space for the station labels in the drawing.

In this thesis, we first use criteria C1, C2, C3, C4 and C5 in our first Grid Graph approach and then we abandon criteria C1 but still keep the others in our second and third approach.

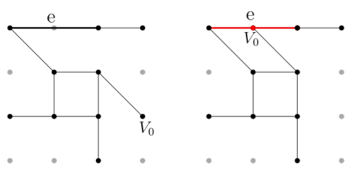
2.2.2 Preserve the topology

The good map design criteria C2 in Section 2.2.1 mentions that the topology of the map must be preserved, which is the basic principle of metro map optimization. Keeping the topology means that no new intersections can be introduced and the counterclockwise order of the adjacent edges of each point remains unchanged.

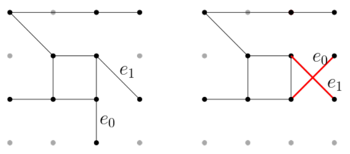
Stott [25] mentioned three cases that can introduce new intersections, which are vertex-vertex overlap, vertex-edge overlap and edge-edge intersect or overlap. Figure 2.4 is an example to show the three cases. And for edge ordering, all incident edges of each vertex are sorted counterclockwise, and the edge ordering of each vertex have to remain unchanged before and after the optimization. Figure 2.5 is an example to show the topology is broken by changing edge ordering of a vertex.



(a) Vertex and vertex overlap.



(b) Vertex and edge overlap.



(c) Edge and edge intersect.

Figure 2.4: An example to show the three cases that introducing the intersections will break the topology. (a): Vertex V_0 will be overlapped with vertex V_1 after moving. (b): Vertex V_0 will overlap with edge e after moving. (c): Edge e_0 will have a intersection with edge e_1 after moving.

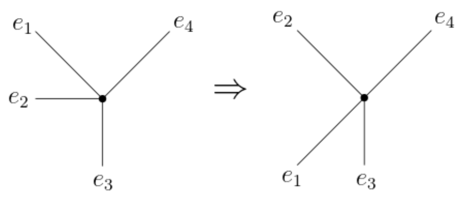


Figure 2.5: An example to show the topology is broken by changing edge ordering of a vertex. The order on the left is $e_1e_2e_3e_4$, but the order on the right becomes $e_1e_3e_4e_4$ after moving, so the topology is destroyed.

2.2.3 Force-directed graph drawing

The force-directed spring model is an excellent model for graph layout problems derived from the physical model, which was proposed by Eades [17] in 1984. Here is a quotation from Eades's explanation:

> The basic idea is as follows (see Figure 2.6). To embed [lay out] a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system... The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state.

Every vertex is subjected to an attractive force and a repulsive force. Eades designed his own formula to calculate the forces. The attractive force is only caused by connected neighbour vertices and the formula is in terms of the logarithm of the distance d between the vertices:

$$f_a = c_1 \log \frac{d}{c_2} \tag{2.1}$$

The repulsive force exists between every pair of vertices, and the formula uses an inverse square law:

$$f_r = \frac{c_3}{\sqrt{d}} \tag{2.2}$$

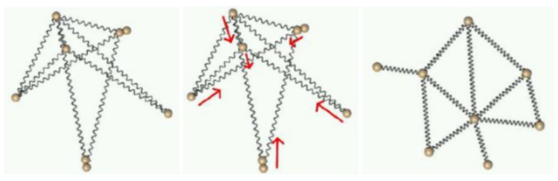


Figure 2.6: An example to show the force-directed spring model. Figure from Kobourov's [20] paper.

Finally when the system reaches dynamic balance, the energy of the whole system is minimum, and the length of each edge is roughly equal to achieve the purpose of a balanced distribution of vertices. Figure 2.7 is an example to show how force-directed algorithm works.

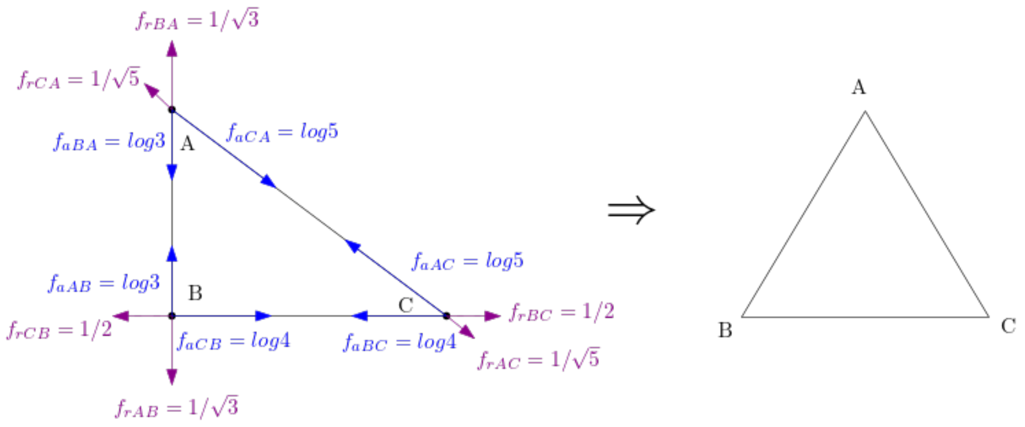


Figure 2.7: An example to show how force-directed algorithm works. The triangle on the left has sides $AB = 3$, $BC = 4$, and $AC = 5$. Each vertex is repulsive and attractive to the other two vertices. For example vertex A, the attractive force from vertex B is $f_{aBA} = log3$ and the repulsive force is $f_{rBA} = 1/\sqrt{3}$. Attractive force is proportional to distance, and repulsive force is inversely proportional to distance. Thus, this triangle will eventually become the equilateral triangle on the right.

Later in 1991, Fruchterman et al. [18] improved Eades's algorithm by adding a method to calculating the ideal distance between vertices. They expected that if the vertices are evenly distributed across the canvas, then the distance between them should be k:

$$k = c\sqrt{\frac{wl}{n}} \tag{2.3}$$

where w and l are the width and length of the canvas, n is the number of vertices, and c is a constant. Then the attractive force and repulsive force can be calculated by k:

$$f_r = -\frac{k^2}{d} \tag{2.4}$$

$$f_a = \frac{d^2}{k} \tag{2.5}$$

Chapter 3

Related Work

In this chapter, we will give an overview of the previous related works on automatic metro-map generation.

Metro map optimization developed from schematization of geographic maps, with additional constraints added. Barkowsky et al. [12] are the first to attempt to automate the process of drawing schematic geographic maps. They apply their algorithm not only to geographic maps, but also to metro maps. Their algorithm focuses on the line simplification of polygons named discrete curve evolution. This algorithm can simplify real geographic curve lines into simple abstract geometry. The algorithm preserves the topology of the input map but it does not deal with the crowded downtown area, and the edge directions are arbitrary.

Avelar and Müller [11] propose an algorithm to automatically generate schematic metro maps with octilinear line segments. They test it with a street map. For each vertex in the map, they calculate a new position iteratively. The new position depends on the neighbourhood of the vertex such that edges approach the expected directions. Besides, they set a maximum length for each segment and a minimum distance between different segments, which makes the map more evenly distributed. Although some lines are not completely octilinear, the readability is greatly improved.

Later, several researches specifically focus on the automatic generation of metro maps. Hong et al. [19] define a set of aesthetic criteria for good metro map layouts. They use a force-directed approach such that only vertices that are neighbours attract each other, but all vertices repel each other. Thus, the additional magnetic forces drag edges towards the closest octilinear direction. In addition, they use a preprocessing step to accelerate the algorithm. Chains of degree-2 vertices are collapsed into one edge and will be re-inserted after computing the simplified layout. This data-reduction step reduces the running time considerably and in our face-based approach, we also take advantage of this idea of re-inserting the degree-2 vertices. The algorithm can improve the readability of metro maps and the edge lengths are more even. But the metro map layout is not perfect octilinearity.

Stott and Rodgers[27] [26] use the multi-criteria optimization approach with a hill climber. Each criterion is weighted and vertices are moved to a new place only when the total of the weighted criteria is reduced. Their criteria include edge intersections, octilinear layout, the length of edges, the angular resolution and the straightness of metro lines. In each iteration, an alternative position is checked for each vertex to improve the quality of the layout. The alternative positions of a vertex is within a circle and the radius of the circle shrinks with each iteration. They also use the same method as Hong et al. to collapse and re-insert degree-2 vertices to reduce the running time. Multi-criteria optimization also does not guarantee perfect octlinearity but the edge lengths are quite uniform. However, the biggest problem of this method is the existence of local minima of the target function in some examples, which is disappointing.

Nöllenburg and Wolff[22] [23] describe a method named mixed-integer linear programming(MIP). This method extends linear programming by introducing the notion of constraining variables to be within certain discrete integer ranges. They define a set of hard and soft constraints. The hard constraints are compulsory while the soft constraints should be approximated as tightly as possible. These soft constraints are used to measure the quality of a metro map. The hard constraints include octilinear layout, thus MIP is the first method that ensures perfect octilinearity. Besides, since MIP does optimization globally, it does not have local minima problem. However, the running time is not polynomial in the worst-case. Thus, when the input scale is large, the algorithm will run for a very long time.

Lutz [21] introduces an efficient algorithm in his Master's thesis. His algorithm can transform a geometric graph into cartogram with given edge lengths and distort edge directions as little as possible. He applies his algorithm to drawing metro maps and the running time is very short. His algorithm uses a method called integer linear programming(ILP), which is based on linear least-squares optimisation. ILP also has soft constraints and Lutz linearises these constraints to simplify the problem. He realizes this by rotating the coordinate system and breaking the angular error down into parallel and vertical components. He defines that a direction error of $\pi/4$ radians is equally as bad as being too long or too short by 50%. Linearization of constraints can greatly improves the efficiency of the algorithm, so they can apply their algorithm in drawing travel-time metro maps.

Recently, there are also some researches on automatically optimizing metro maps. In 2020, Wu et al.[28] summarized the existing research on transit map layout, and analyzed it from the perspectives of designers, machines, and users, and provided ideas for future research directions. Bast et al. [13] proposed a method to automatically generate octilinear metro maps where the number of bends will be optimal, and the edges can be bent arbitrarily. Later in 2021, they improved the algorithm to make it more general [14], automatically generating all kinds of layouts, including octalinear, hexalinear and orthogonal.

All of the above related works are vertex-based, which distort the metro map by directly moving the vertices and edges. Now our project will introduce a completely different face-based approach. We mainly change the size and shape of the faces between the metro lines to make the metro map evenly distributed and improve the readability.

Chapter 4

Approach 1: Grid Map

We have tried three different approaches and experimented with each approach. Our initial two approaches ultimately failed to give satisfactory results. This chapter discusses our ideas for the first approach, including why we choose to use a grid map, how we optimize the size and shape of the face with the help of the grid map and why this approach does not work.

4.1 Introduction

In this section, we give a brief introduction of our grid map approach. We discuss our decisions in the design stage and give the pseudo-code for the optimization stage. Then we give an overview of this chapter.

Advantages of grid map

A grid map is an integer square grid and vertices can only be put on the grid intersections (black dots), as is shown in Figure 4.1a. For convenience, we omit the lines between the grids and keep only the grid intersections. The distance between two adjacent grid intersections is g and we call g the edge length of the grid map. In the design stage of this approach, we embed our input metro map into a grid map, which means each vertex will be pulled to an integer grid intersection. We use this approach because the grid map has many advantages. Firstly, we decide to use the same design criteria used by the vast majority of metro maps mentioned in section 2.2, i.e. use octilinear layout, keep topology, smooth bends, evenly distribute stations and leave enough space for labels. Thus, we choose to use the grid map because we think it is convenient to control the

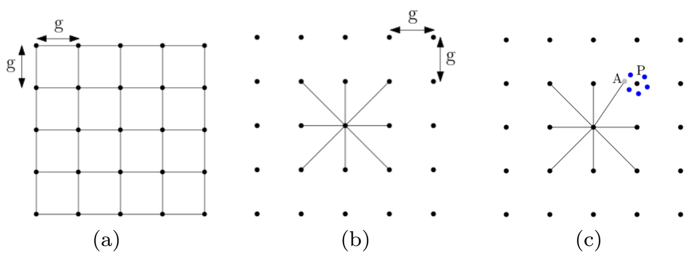


Figure 4.1: An example to show the advantages of grid map. (a): A grid map is an integer square grid and vertices can only be put on the grid intersections. (b): The convenience of grid map in controlling the octilinear layout and edge length. (c): A grid map can significantly reduce the number of potential positions for vertices. Vertex A can be placed directly at point P, otherwise it has to test other potential candidates such as the blue points.

octilinear layout (**C1**) and edge length (**C4**) with fixed and evenly spaced grids. As is shown in Figure 4.1b, edges can easily be placed horizontally, vertically or diagonally in the grid map and edge lengths are exactly the distance g between grids in the horizontal or vertical direction (and $\sqrt{2}g$ in diagonal direction). Another reason we choose to use the grid map is because the grid intersections can significantly reduce the number of potential positions for vertices. As is shown in Figure 4.1c, vertices can only be placed at grid intersections, which can improve our algorithms efficiency greatly.

Embed in the grid map

As a first step, we have to embed our metro map into the grid map. This is because our input data is the exact geographical positions, but our grid map only has integer coordinates. Figure 4.2 is an example to show embedding the metro map to integer square grids.

The embed method we use is the same as Stott and Rodgers [27]. We first drag every vertex to its nearest grid intersection, as is shown in Figure 4.2. If more than one vertex shares the same nearest grid intersection, we move all the redundant vertices to their second nearest vacant grid intersections. Figure 4.3 is an example to show this. If there are still conflicts for the grid intersections, that means the grid is too sparse and we should reduce the grid size. As is shown in Figure 4.4, shortening the grid length g will increase the number of grid intersections, which can reduce the conflicts.

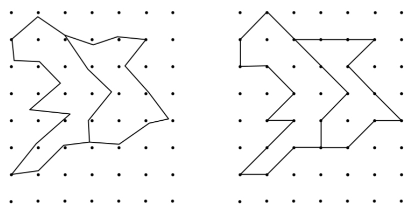


Figure 4.2: An example to show how to embed a map with accurate geographical float value on an integer square grid map.

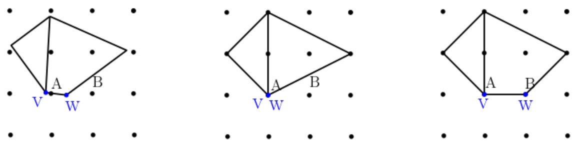


Figure 4.3: An example to show how to deal with the conflicts for the grid intersections. Vertex v and vertex w both have the grid intersection A as the nearest. But they can not both move to A, because there would be overlap. Thus, we can move vertex v to A and then move vertex w to its second near vacant grid intersection, which is B.

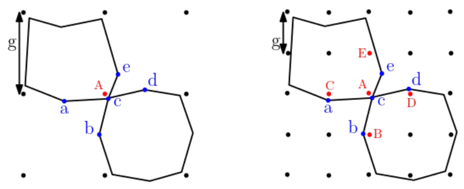


Figure 4.4: An example to show shortening the grid length g can reduce the conflicts. The vertices a,b,c,d,e on the left are all nearest to A, and there are not enough vacant grid intersections. Then we shortening the grid length to increase the number of vacant grid intersections, so that the five blue vertices can move respectively to the five red intersections.

The pseudo-code of the first optimization algorithm

Algorithm 1: Approach 1: grid maps

```
Input: The routes of the metro map and the original geographic coordinates of all
       stations on the routes.
Output: The optimized station coordinates.
1  Embedded the original geographic metro map into the grid map;
2  for each face do
3      Calculate the roundness of the face (Section 4.2);
4      while roundness <threshold do
5          Calculate the centroid $C$ of the face (Section 4.3);
6          Define the ideal radius $r$ of a circle (Section 4.4);
7          Draw a circle with $C$ as the center and $r$ as the radius;
8          for each vertex on the face do
9              move the vertex towards to the centroid if it is outside the circle;
10             move the vertex away from the centroid if it is inside the circle;
11             (Section 4.5 and 4.6);
```

After the embedding step, we optimize the faces of the metro map one by one. In the optimization stage we want to adjust the shape and size of the faces so that the map meets as many of the criteria as possible. Here we give an overview of our algorithm and we will give more details as well as our motivation in the later sections. The pseudo-code of our grid maps approach is shown in Algorithm 1. Figure 4.5 is a sketch of the process. For each face, we want to increase its roundness as much as possible because the closer the face is to the circle, the larger the space between the routes and the less likely it is to be crowded. If the roundness is smaller than our threshold, we first calculate the centroid of the face and draw a circle with the centroid as the center. We give a definition of the optimal radius of the circle and vertices on the circle are considered to have the optimal distance from the centroid. The vertices inside the circle are too close to the centroid, resulting in the small roundness. So the inside vertices should be moved away from the centroid. Similarly, the vertices outside the circle should move toward the centroid and finally all the vertices fall near the circle. After all the vertices are processed once, the roundness is calculated again. If the roundness is still less than the threshold, the algorithm is repeated until the roundness is greater than the threshold.

In the following sections, we will step by step give the motivation and algorithmic details for each of our steps. Section 4.2 talks about why we want to increase the roundness and Section 4.3 shows why we need the centroid of a face and how to calculate it. Then we describe how to calculate the radius for the circle in Section 4.4 and show how to increase the roundness of a face in Section 4.5. Section 4.6 is about how to find a target grid for each vertex and we will show the results of our algorithm in Section 4.7.

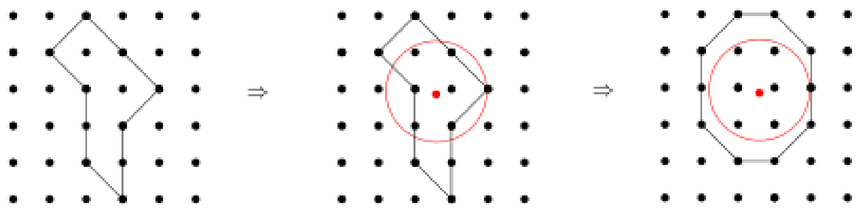


Figure 4.5: The algorithmic process of the Grid Approach.

4.2 Calculate the roundness of a face

In Section 2.2.1, criteria **C5** states that the layout should leave enough space for the labels in the drawing. Considering that among all geometric figures with the same perimeter, the area of the circle is the largest, we decide to use the roundness as one of the standards to measure a face. The greater the roundness of a face, the closer the face is to a circle, so our algorithm should maximize the roundness of each face as much as possible to leave enough space for the labels. Another benefit of increasing the roundness of the faces is that it increases the distance between the close routes, which can avoid some sharp bends and prevent clusters of stations. As Figure 4.6 shows, when the roundness of the face increases, the distance between two close routes increases and the space left for the label becomes larger. This can reduce the clusters and overlaps and can make the routes more smooth.

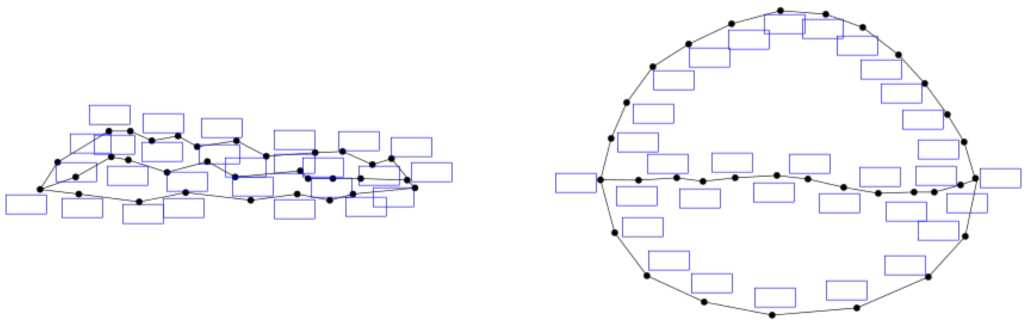


Figure 4.6: An example to show why we use the roundness to measure a face. The map on the left has small roundness. Vertices are clustered, labels are overlapped, bends of the routes are very sharp. However, when the roundness of the face is increased on the right, the clusters and overlaps are reduced and routes are more smooth.

Roundness is the area of a face over a circle that has the same perimeter as the face, so we can calculate roundness as follows:

$$roundness = \frac{area(Face)}{area(Circle)} = \frac{area(Face)}{\pi(\frac{Perimeter(Circle)}{2\pi})^2} = \frac{4\pi * area(Face)}{(Perimeter(Face))^2}$$

4.3 Calculate the centroid of a face

We observe that for a circle, its center is exactly its centroid, and all vertices on the circle are at the same distance from the centroid. Conversely, when the shape of a face is irregular, the distance between the vertices on the face and the centroid will be very unbalanced. As shown in Figure 4.7, for the face on the right, the distance from the vertices on the face to its centroid is not balanced, so its roundness is smaller than the circle on the left. We assume, then, that the greater the roundness of a face, the smaller the difference in distance between vertices on the face and its centroid. Thus, our algorithm should aim to modify the shape of the face by reducing the distance difference between vertices on the face and the centroid to increase the roundness.

The centroid of a polygon face can be calculated as follows:
First divide the face into triangles, and calculate the centroid and area of every triangle:

$$centroid(\frac{x_0 + x_1 + x_2}{3}, \frac{y_0 + y_1 + y_2}{3}) \tag{4.1}$$

$$S = \frac{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)}{2} \tag{4.2}$$

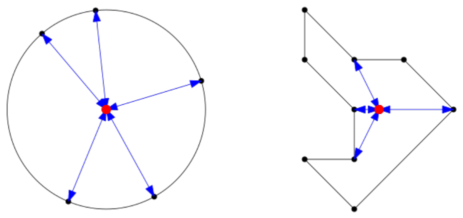


Figure 4.7: An example to show smaller distance difference between vertices and centroid leads to greater roundness face.

Then the centroid of the face is: $(X = \frac{\sum_{i=1}^{n}(x_i s_i)}{\sum_{i=1}^{n} s_i}, Y = \frac{\sum_{i=1}^{n}(y_i s_i)}{\sum_{i=1}^{n} s_i})$,

where (x_i, y_i) is the centroid of the ith triangle, and s_i is the area of the ith triangle.

4.4 Calculate the radius for *FaceCircle* of a face

To reduce the distance difference between vertices on the face and the centroid, we draw a circle with the centroid as the center, called *FaceCircle*, like Figure 4.8 shows. Vertices on the circle are regarded to have the optimal distance to the centroid. Thus, vertices inside the *FaceCircle* should be moved further away from the centroid to increase their distance, and vertices outside the *FaceCircle* should be moved closer to the centroid to reduce their distance. Ideally, all vertices end up on the *FaceCircle*, have the same distance from the centroid. Thus, defining the radius of *FaceCircle* is very important.

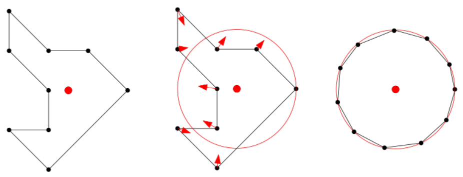


Figure 4.8: An example to show *FaceCircle*. The circle take the centroid as the center. Vertices inside the circle will be moved further away from the centroid and vertices outside the circle will be moved closer to the centroid. Ideally, all vertices will end up on the *FaceCircle*, and have the same distance from the centroid.

We calculate the radius of the *FaceCircle* through a regular polygon and the idea is as follows. Our design criteria specify that our edges are straight lines (Octilinear layout), so each face cannot be a true circle, but rather as a polygon as close to the *FaceCircle* as possible (The right-most one in Figure 4.8). Of all polygons, the regular polygon with all vertices exactly on the *FaceCircle* is closest to the *FaceCircle* and has the maximum roundness and area. It also has uniform edge length and has a regular shape. As shown in Figure 4.9, the red circle has the *FaceCircle* of the

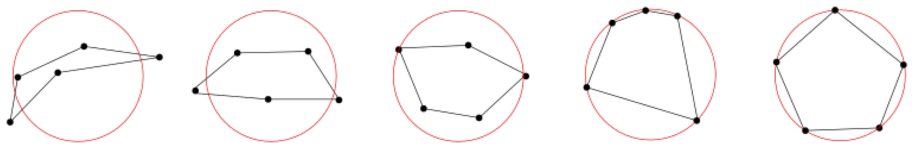


Figure 4.9: An example to show the regular polygon with all vertices on the circle has the maximum roundness and area, as well as the uniform edge length and regular shape.

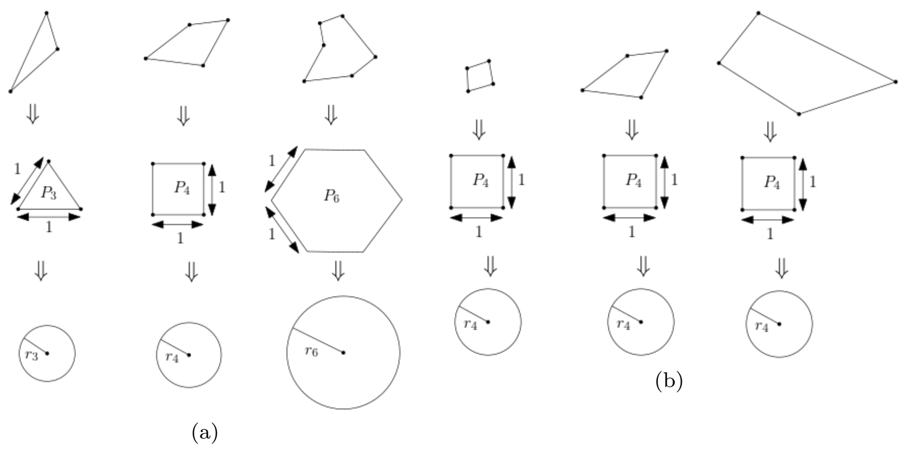


Figure 4.10: An example to show the calculation for the radius of *FaceCircle*. (a): Calculate the radius of *FaceCirce* for triangles, quads, and hexagons. For each face, we calculate a regular polygon with the same number of the face and take the edge length to be l. The circumcircle of the regular polygon is the the *FaceCircle*. (b): The radius of *FaceCircle* is only determined by the number of vertices on the face and it does not matter how big the face was. For these three quadrangles, their regular polygons are squares of edge length l, so their optimal area and *FaceCircle* are exactly the same.

face and vertices on the circle have the optimal distance to the centroid. The right-most regular polygon is the maximum roundness for a face with five vertices. For every face we can calculate the regular polygon that is exactly on the *FaceCircle*, as is shown in Figure 4.10a. The *FaceCircle* is the circumcircle of the regular polygon and as long as we know the edge length of the regular polygon, we can compute the radius of *FaceCircle*.

We define the edge length of a regular polygon as l and the radius of the *FaceCircle* can be calculated as Figure 4.11 shows. The edge length l of the regular polygon can determines the optimal distance between the centroid and the vertices and thus influences the size and shape of the face. We will discuss this later and will compare the effects of different l to the final metro map.

We define that the optimal area of a face is proportional to the number of vertices on the face. Faces with more vertices need more area for labels and more space to avoid clustering. Therefore, such an area distribution can make the metro map more evenly balanced. Figure 4.12 is an example of the proportional distribution. The area of regular polygon is determined by the number of vertices on the face, so the faces with the same number of vertices will have the same radius for *FaceCircle*, no matter how big the faces' area was before, as Figure 4.10b shows.

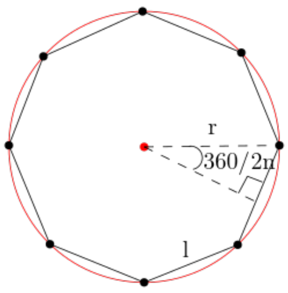


Figure 4.11: An example to show how to calculate the circumcircle of a regular polygon. The edge length is l and the central angle is $360/2n$, where n is the number of the vertices on the face.

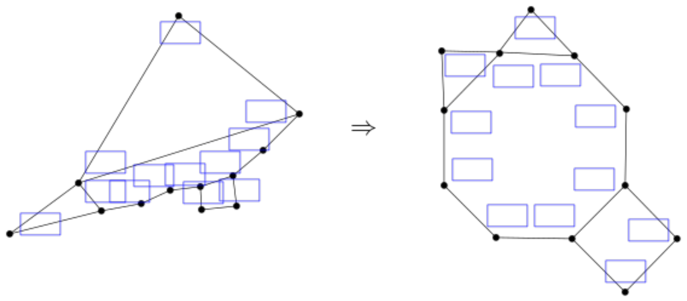


Figure 4.12: An example to show the area distribution is proportional to the number of vertices on the face.

4.5 Increase the roundness

To increase the roundness, we have to reduce the distance difference from vertices to the centroid. We have tried two methods to improve the roundness.

Method1: Use Force-directed method

In this method, the forces on the vertices are exactly along the radius. To be specific, vertices inside the $FaceCircle$ are repulsed away from the centroid, and vertices outside the $FaceCircle$ are attracted toward to the centroid, as is shown in Figure 4.13. The repulsive and attractive forces can be calculated with the formulae in Section 2.2.3. The attractive force is $f_a = \log d$ and the repulsive force is $f_r = \dfrac{1}{\sqrt{d}}$, where d is the distance between the vertex and the centroid. According to the formulae, the closer a vertex is to the centroid inside the circle, the more repulsive force it will receive, which pushes it farther away from the centroid; and the farther away a vertex is from the centroid outside the circle, the greater attractive force it will receive, which makes it be attracted closer to the centroid. This is exactly what we want: to balance the distance between vertices on the circle and the centroid.

However, we find this method sometimes gives unexpected results. Figure 4.14 shows the unexpected situation, since vertices A and B can only be attracted or repelled along the radius, the final result has intersections that break the topology. Besides, it has very sharp bends, violating the criteria C3 in Chapter 2.2, which says the bends should be as smooth as possible. This restriction on the direction of movement also causes clusters, which prevents vertices from being evenly distributed on the circle.

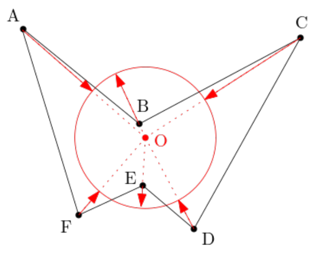


Figure 4.13: An example to show Force-directed method to increase roundness. Vertices inside the $FaceCircle$ (B and E) are repulsed in the opposite direction of the centroid, and vertices outside the $FaceCircle$ (A,C,D,F) are attracted toward the direction of the centroid. The repulsive force is inversely proportional to the distance from the vertex to the centroid (B is closer to O than E so B will be repulsed further). The attractive force is proportional to the distance from the vertex to the centroid (A is further to O than F so A will be attracted furthur than F).

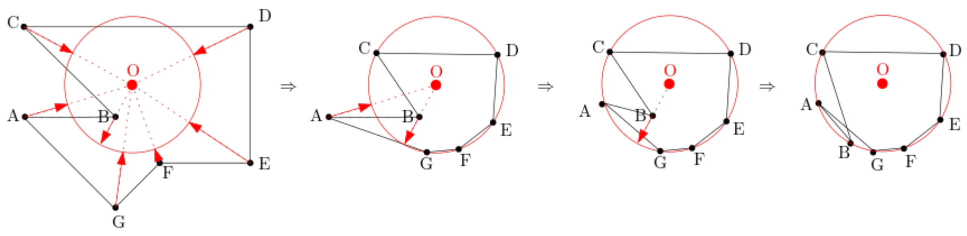


Figure 4.14: An example to show the irrationality of force-directed method in increasing the roundness of a face. The four figures are: the original face, the face that has been processed at all vertices except A and B, the face with A processed, and the final face with B processed. A and B can only move along the radius, causing inevitable intersections or sharp bends. In addition, the upper half of the circle is sparsely distributed, while the lower half of the circle is clustered

Method2: Divide face into left and right chains

Then we propose another way to increase the roundness of the face. Instead of restricting the movement of vertices in the direction of the radius, the face is divided into left and right chains, with each vertex on the chains has five moving directions. As is shown in Figure 4.15a, we first find the highest vertex and the lowest vertex of the face, use these as the dividing points to divide the face into two chains of blue and green. Every vertex on left blue chain inside the $FaceCircle$ has five directions, that is, *top*, *topLeft*, $left$, *bottomLeft*, *bottom*; every vertex on right green chain inside the $FaceCircle$ also has five directions, that is, *top*, *topRight*, *right*, *bottomRight*, *bottom*. Vertices that are outside $FaceCircle$ have the opposite five directions. Each vertex can only move one grid distance at a time or stay unmoved, which means that each vertex has at most six positions to choose. Figure 4.15b is an example to show the candidate positions.

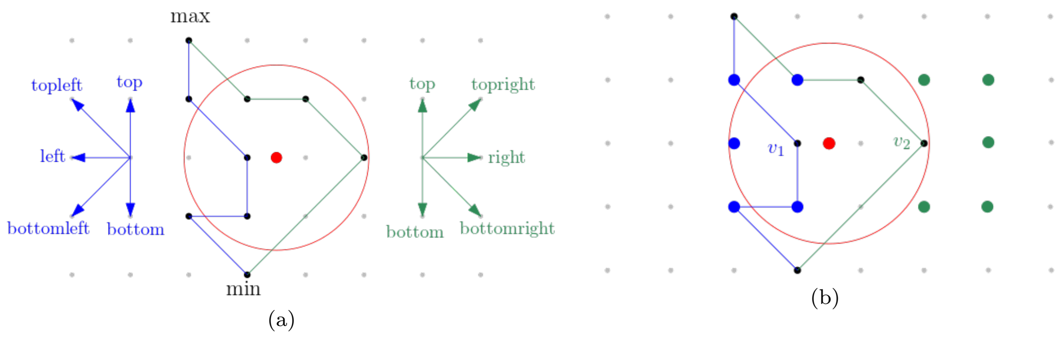


Figure 4.15: Divide face into left and right chains. (a): An example to show how to divide face into chains and the directions for vertices on chains. (b): An example to show five candidate positions that a vertex can be moved to. v_1 is on left chain and inside the $FaceCircle$, and its five candidate positions are marked with blue dots, but it can only choose $left$ position because there are already vertices on other candidate positions. v_2 is on right chain and inside the $FaceCircle$, and its five candidate positions are marked with green dots, and it can be moved to any of the candidate positions.

4.6 Find a target grid

Although every vertex has five candidate positions, not every candidate position is available. We must ensure that the metro map is octilinear and the topology is unchanged during the movement. Thus, every time we want to move a vertex to a candidate position, we must check whether the angle between all adjacent edges of this vertex and the x-axis is an integer multiple of 45 degrees. If not, this candidate location is not available. For example in Figure 4.16, vertex v can not be moved to its candidate location *bottom* because after moving, octilinear layout will be destroyed. Likewise, we have to check the topology if we want to move a vertex to its candidate position. If there exist new intersections or if it changes the ordering of edges (see Section 2.2.3), the candidate position also has to be abandoned.

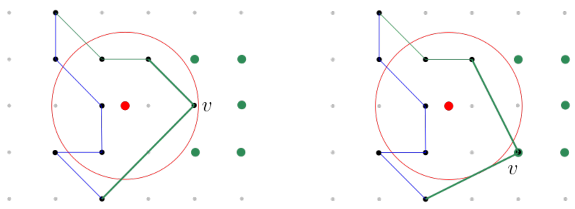
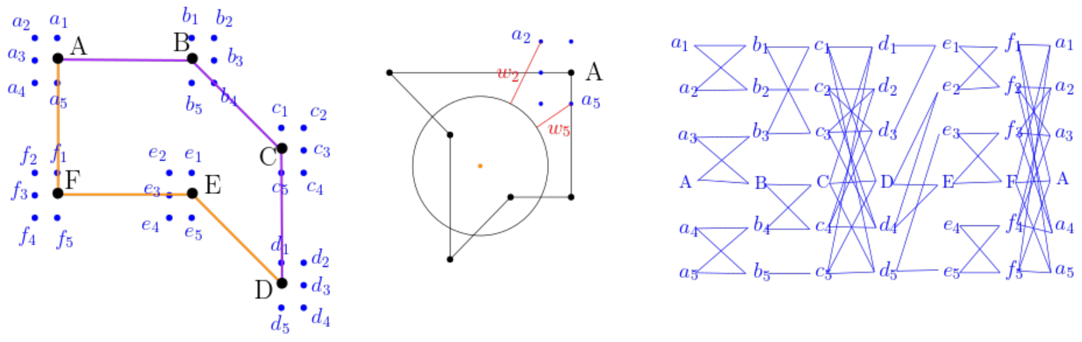


Figure 4.16: An example to show candidate positions where the angles between the adjacent edges and the x-axis is not an integer multiple of 45 degrees are not available.

For a single face, we want to update the positions for all vertices simultaneously. We use the shortest-path method to realize this and the idea is shown in Figure 4.17. For each vertex, we draw its six candidate positions and list them in a column. Each candidate position has a weight and the weight is equal to the distance from candidate position to the $FaceCircle$. The columns for the two adjacent vertices in the face are listed next to each other. Then for each candidate position of a vertex, check which candidate positions of its neighbor's are octilinear with it and do not break the topology. If such a candidate position exists, the corresponding points are connected by a line between their candidate position columns. After all the vertices on the face go through this step, we will obtain the shortest-path graph on the right.

Then we can directly get the new shape for the face. To be specific, we randomly start from a candidate position in the shortest-path graph, and we find a shortest path that gets back to this candidate position again. All the candidate positions on the path are the new locations for the vertices of the face. Then we can obtain a new shape of the face. We only choose shapes that make the roundness of the face larger. If this path does not increase roundness, we find another path. Then we update all the vertices of a single face simultaneously. Figure 4.18 is an examples to show how to get a new shape from the shortest-path graph.



a_2 a_1 a_3 A a_4 a_5
b_1 b_2 B b_3 b_5 b_4
c_1 c_2 C c_3 c_5 c_4
f_2 f_1 F f_3 f_4 f_5
e_2 e_1 e_3 E e_4 e_5
d_1 d_2 D d_3 d_5 d_4
a_2 w_2 A a_5 w_5
a_1 b_1 c_1 d_1 e_1 f_1 a_1
a_2 b_2 c_2 d_2 e_2 f_2 a_2
a_3 b_3 c_3 d_3 e_3 f_3 a_3
A B C D E F A
a_4 b_4 c_4 d_4 e_4 f_4 a_4
a_5 b_5 c_5 d_5 e_5 f_5 a_5

Figure 4.17: An example to show the shortest-path method that can update the positions for all vertices on a single face simultaneously. Suppose that the face is too small and all the vertices are inside the $FaceCircle$. a_1, a_2, a_3, a_4, a_5 are the candidate positions for vertex A and are listed in a column on the right figure. Then there are the B, C, D, E, F candidate position columns. Then we check for the octilinear. For example, candidate position a_1 of vertex A can only go to candidate positions b_1 and b_2 of vertex B and the edges between them are still octilinear. Then we draw a line between a_1b_1 and a_1b_2. For the figure in the middle, the weight for candidate position a_2 is w_2 and candidate position a_5 has weight w_5.

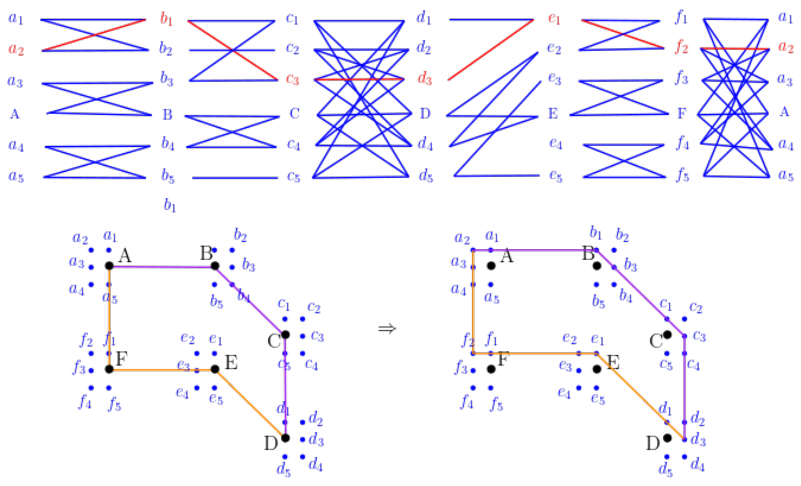


Figure 4.18: An axample to show obtaining a new shape from the shortest-path graph. For example, we start form a_2, and then find a path $a_2b_1c_3d_3e_1f_2a_2$ that back to a_2, as is shown in the red line. Then we can update vertices A, B, C, D, E, F to $a_2, b_1, c_3, d_3, e_1, f_2$ respectively and get a new shape for the face.

However, when there are multi faces in a metro map, the shortest-path graph can not be used anymore. This may break the octilinearity of neighbour faces, as is shown in Figure 4.19. Thus, in our implementation, we can not update all the vertices of a face at the same time, instead we have to move vertices one by one to check the octilinearity and topology.

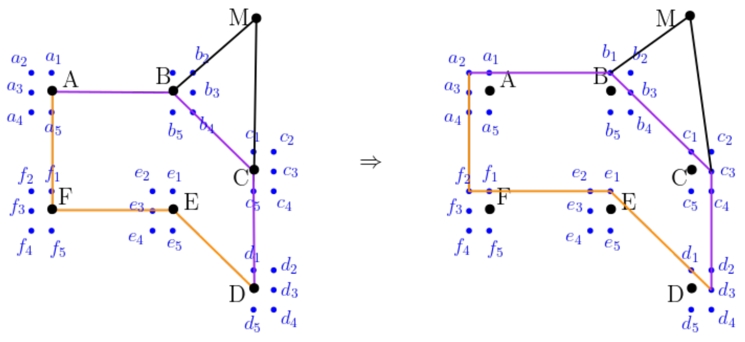


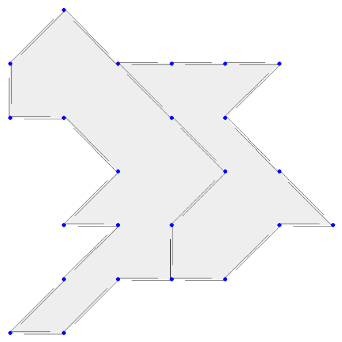
Figure 4.19: An axample to show shortest-path graph can not be used for multi faces. After the left face changes its shape according to the shortest-path graph, the edges of the triangle above is no longer octilinear.

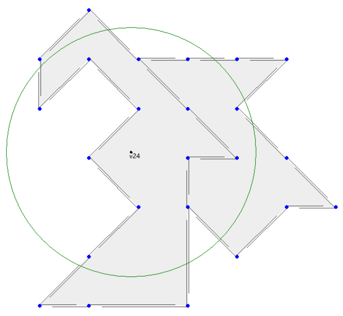
4.7 Results

For this method, we did not experiment on a real map, but used a demo graph containing only two faces and skipped the embed step to see the effect. We chose the edge length of the regular polygon to be one. We want the roundness of each face to be as large as possible, but it is impossible to achieve maximum roundness due to the influence of neighboring faces and the octilinear restriction. Besides, the fewer vertices a face has, the smaller the maximum roundness a face can achieve. The face with the fewest vertices is the triangle, and the maximum roundness of a positive triangle with an edge length of one is $\sqrt{3}\pi/9(\approx 0.6)$. Therefore, we define that 0.5 is a threshold for roundness, and we only adjust the faces with roundness less than 0.5. For the faces with roundness greater than 0.5, we think they have left enough space for labels and the cluster is not serious.

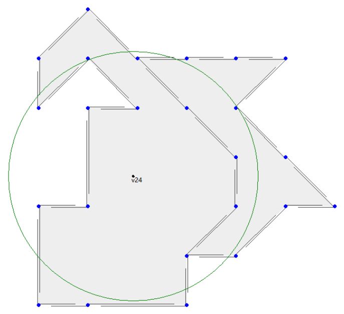
The process of changing the shape of the map using the first approach is shown in the Figure 4.20. Figure 4.20a is the original input. First deal with the left face and in each iteration we get a new position for each vertices on both chains. After calculation, the roundness of this face is smaller than 0.5, so divide the face into left and right chains and check the available candidate positions for each vertex. After this loop, the map looks like Figure 4.20b and we get a new left face and calculate the roundness again. The roundness is still less than 0.5, so we repeat the progress and get Figure 4.20c, Figure 4.20d and Figure 4.20e. Now the roundness of left face is larger than the threshold, so we move to next face. Figure 4.20f shows the algorithm begins to deal with the right face, and Figure 4.20g is the final result of the whole map.

We can see that it is true that the roundness of the faces can become larger, the shapes of the faces can become more regular, the edge length are even, and the layout is octilinear, which

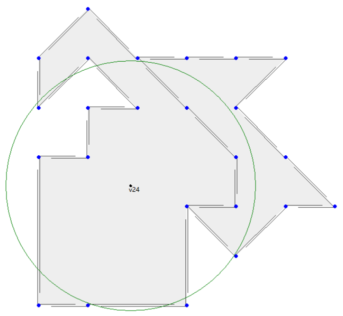

(a) Original shape.



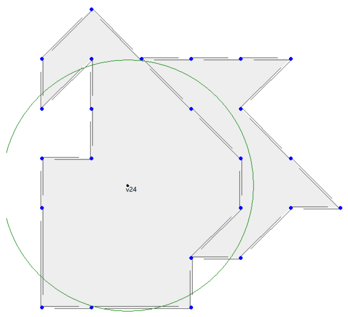
(b) Left face one step.



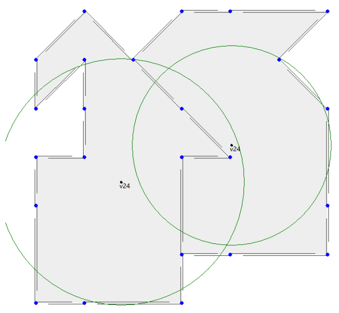
(c) Left face two step.



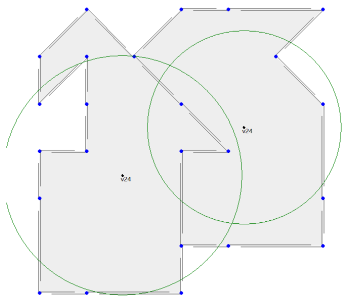
(d) Left face three step.



(e) Left face four step.



(f) Right face one step.



(g) Right face two step.

Figure 4.20: Examples to show the process of changing the shape of the map during processing. The left face is processed first, and if the roundness is less than the threshold the algorithm will be repeated until the roundness is greater than the threshold. Then the right face is processed.

suits our purposes. However, this algorithm has fatal problems. First, because the algorithm deals with faces one by one, the face that has been processed will be affected by its neighbors later, resulting in invalid processing. For example, in Figure 4.20e, the roundness for the left face meets the threshold, but when dealing with the right face in Figure 4.20f, their common edges are pushed back again, causing the left face to become smaller again.

Second, we regard the regular polygon of every face to be its optimal shape and size, but the octilinear layout makes it impossible for the face to be a regular polygon. The distance between two grid is g in horizontal and vertical directions but $\sqrt{2}g$ in diagonal direction. It is not possible that all vertices of a regular polygon are on the grid intersections. In other words, it is impossible for the face to reach maximum roundness and the edges cannot be equal in length. Figure 4.21 is an example for the first problem.

Third, we set the threshold of the roundness to be 0.5, which is not general for all the faces. For faces with 3 vertices, the maximum roundness is 0.6 so 0.5 is a good threshold. However, for faces with more vertices, their roundness maybe larger than 0.5 but still do not have the optimal shape. They may have sharp bends or very long edges. Figure 4.22 is an example of this problem.

Fourth, this algorithm only works for some specific faces. The faces in our demo already have octilinear layout, and all edge lengths are exactly the grid length g ($\sqrt{2}g$ in diagonal). This is a very special case and does not represent the faces of the real geographic metro map embedded in the grid map. The edge length of a true embedded face is often very uneven and the layout is not octilinear, as is shown in Figure 4.23. The algorithm does not work for such faces because all candidate positions for all vertices are unavailable. Thus, these irregular faces are never optimized. Besides, the final results are very likely to contain inappropriate bends, such as the upper left corner of Figure 4.20g.

All in all, the first method combined with grids cannot achieve our goal well. So we decided to change our exploration direction.

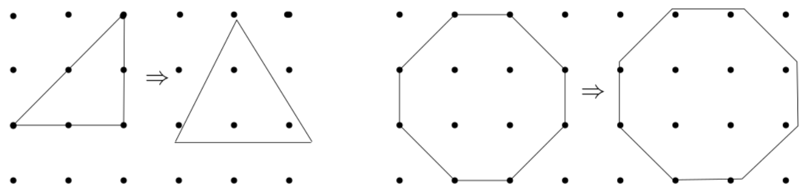


Figure 4.21: An example to show the octilinear layout makes it impossible for the face to be a regular polygon. To the left of the arrow are the faces embedded in the grid map, and to the right of the arrow are the standard regular polygons.

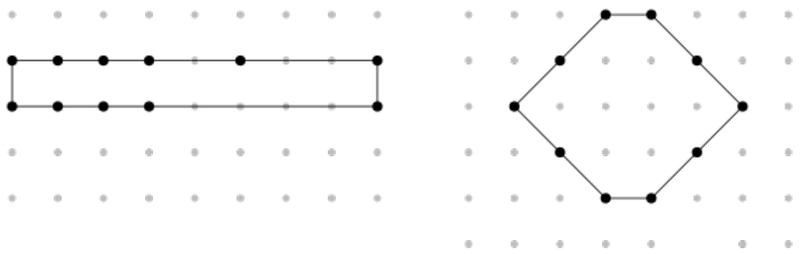


Figure 4.22: An example to show setting the threshold of the roundness to be 0.5 is unreasonable for faces with many vertices. The roundness of this face is already larger than 0.5, so it will not be optimized. But it has unbalanced edges and it can never be optimized to the shape shown on the right.

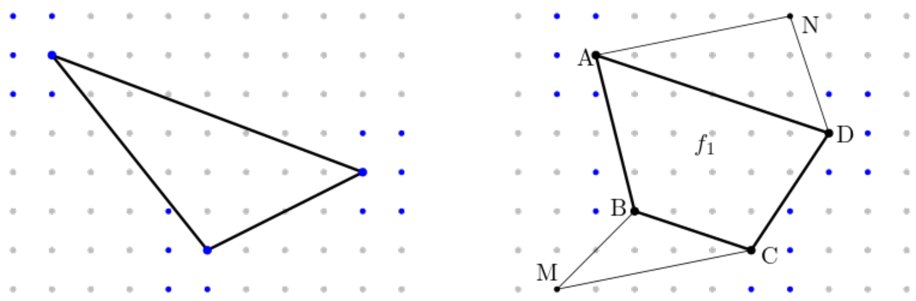


Figure 4.23: An example to show faces that the algorithm can not process. The triangle on the left is too large, but the candidate positions (blue dots) for all vertices are not available because none of these dots can make the face to be octilinear. Same reason for the faces on the right.

Chapter 5

Approach 2: Attach Circle

This chapter discusses our ideas for the second approach. In Section 5.1, we first discuss why we abandon the octilinear layout and the grid map and describe the advantages of our second Attach Circle approach. Then we give the pseudo-code and our general idea of this approach. Next we describe our attach regulation in Section 5.2 and show how we deal with the attachment when there are multiple circles in Section 5.3. In Section 5.4 we show how we preserve the topology of the whole metro map. In Section 5.5 we show the results on real metro maps of this approach and discuss the advantages and disadvantages.

5.1 Introduction

We learn form the first approach that one of the reason for the failure of the first approach is that there are too many restrictions on the location selection of vertices. Vertices can only be put on the intersections of the grid map and the octilinear layout places too many restrictions on the shape of the faces. The horizontal and vertical lengths (g) of the grid are not the same as the diagonal lengths ($\sqrt{2}g$), making it impossible for faces to become perfect regular polygons. Edges of the faces and x-axis angles can only be an integer multiple of 45 degrees, making it impossible for faces to reach a maximum roundness close to the circle. In addition, if none of the five candidate positions for vertices are available, irregular faces will never be optimized. All of these problems are caused by the design criteria C1 (octilinear layout) and contrary to our optimization ideas. We want to smooth the shape of the faces and maximize the roundness of the faces. Therefore, we have to change our design criteria.

Although the octilinear layout is the most common design style for metro maps, but it does not perform well in our 'face-based' algorithm. We observe that with regular faces shape and balanced space distribution, metro maps are also readable and useful. For example, the schematised curvilinear layout 1.4b in Figure 1.4 is much more beautiful and practical than the original curvilinear input 1.4a. There are no clusters of stations and the routes are very smooth. The space distribution is also better and the area of the faces are approximately proportional to the number of vertices on the face. Inspired by this, we think we can use a similar design layout in this approach and we can further optimize the curve edges to straight lines. Therefore, we decide to combine the multilinear and schematised curvilinear layout to replace octilinear layout C1 in Section 2.2. The only difference between our layout and schematised curvilinear is that we still use straight lines to connect stations, as shown in Figure 5.1. Besides, we still keep the criteria C2 to C5 in Section 2.2 for these are the hard requirements of a good metro map. Thus, our design criteria in the second approach are as follows. We combine a schematised curvilinear with multilinear layout; we require the preservation of the topology; we want to reduce bends; we aim to obtain a more even edge length and leave enough space for labels.

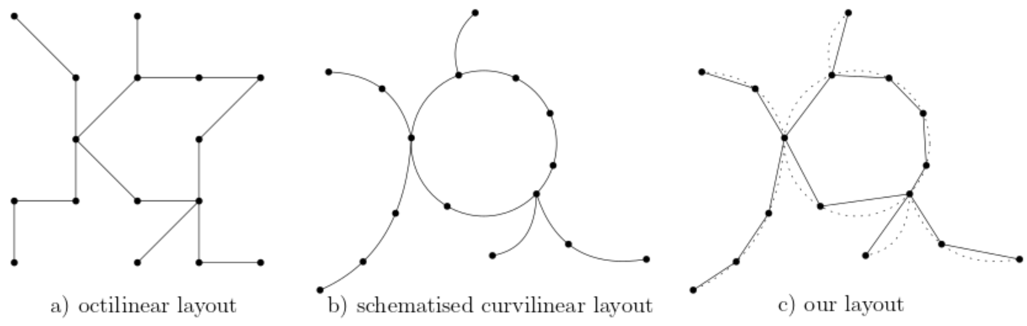


Figure 5.1: An example to show our metro map layout. The overall layout fits schematised curvilinear but we use straight lines to connect stations. Vertices can be moved to any positions and do not have to be on integer grids.

5.1.1 Advantages of Attach Circle approach

From Grid Map approach, we learn that if we deal with only one face at a time, the faces that have already been processed will be affected by its neighbors. Their common edges will be pushed back, making the roundness of the processed face smaller again. So dealing faces one by one does not guarantee the roundness of the processed faces. Thus, we want to come up with an approach that can deal with all the faces in the metro map at the same time. We call it Attach Circle approach and the general idea is shown in Figure 5.2.

The pseudo-code of the second optimization algorithm

Algorithm 2: Approach 2: Attach Circle

```
Input: The routes of the metro map and the original geographic coordinates of all
       stations on the routes.
Output: The optimized station coordinates.
1 for each face do
2     Calculate the centroid and FaceCircle as Grid Map approach;
3     for each vertex on the face do
4         Find the ideal target position on the circle (Section 5.2);
5         Calculate the displacement vector towards the target position (Section 5.3);
6         if Moving the vertex to the new position does not break the topology (Section 5.4)
           then
7             Move each vertex to its target position;
```

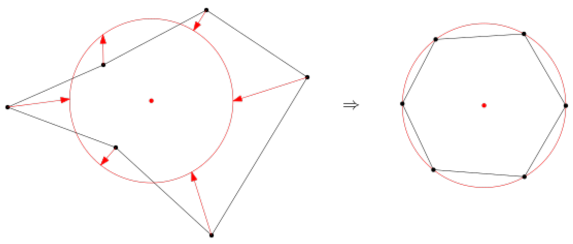


Figure 5.2: The optimization idea of Attach Circle approach. We find target positions on the *FaceCircle* for all vertex and drag them to the circle directly.

In this approach, without the direction restriction of the edges, we can directly move the vertices on the face to the *FaceCircle* and distribute the vertices evenly on the circle. This can make the face to be a perfect regular polygon wuth the largest roundness, which is in line with our optimization ideas. We regard the movement of the vertex as a displacement vector, the original position is the vector tail, the target position is the vector head, and the movement distance is the vector length. Figure 5.3a is an example to explain the displacement vector. We will discuss later how to find the target position. And when a vertex has multiple incident faces, we can directly add up the displacement vectors caused by all faces and obtain the final position of the vertex. Figure 5.3b is an example to show multi faces. Therefore, we can calculate the final positions for all vertices on the map first and then update their coordinates at the same time. The advantage of this approach is that it can prevent the problem of common edges being pushed back and faces can be optimized to be the regular polygons with maximum roundness.

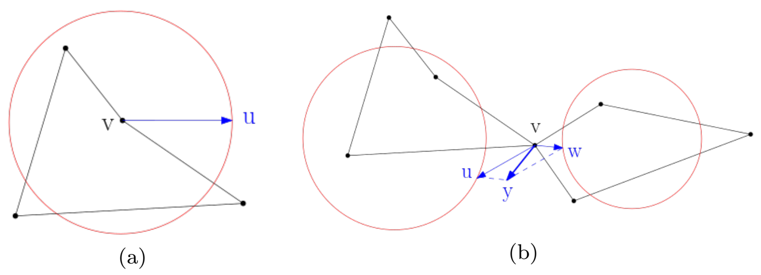


Figure 5.3: Examples to show the displacement vector. (a): Displacement vector for a single circle. If we move vertex v to point u on the circle, we will get a displacement vector $\overrightarrow{vu}$ for vertex v. The original position v is the vector tail, the target position u is the vector head, and the movement distance $|vu|$ is the vector length. (b): Vector addition for multi incident faces. Vertex v has two incident faces so it has two displacement vectors $\overrightarrow{vu}$ and $\overrightarrow{vw}$. We can use vector addition to calculate the final displacement vector $\overrightarrow{vy}$ for vertex v.

5.2 Attach regulation

We still want to make the faces as round as possible but instead of using roundness as the threshold for measuring the face, we expect to directly attach vertices on the face to *FaceCircle*. As is shown in Figure 5.4, we first calculate the regular polygon of the face and draw the *FaceCircle* as in the Grid Map approach. Then we rotate the regular polygon, making one vertex of the regular polygon to overlap with the rightmost point of the *FaceCircle*. The points at which regular polygon and *FaceCircle* overlap are the target positions for the vertices of the face.

Secondly, we attach vertices on the face to target locations on the *FaceCircle*. As Figure 5.5 shows, we attach the vertex with the largest horizontal coordinate on the face to the rightmost

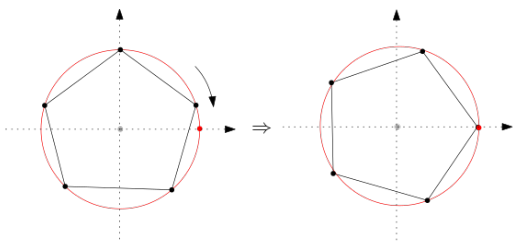


Figure 5.4: An example to show how to find the target positions for each vertex on *FaceCircle*. Rotate the regular polygon that one vertex overlaps with the rightmost point (the red one) of the *FaceCircle*.

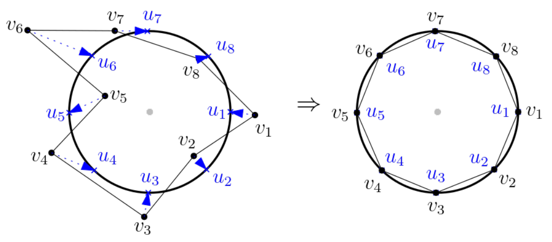


Figure 5.5: An example to show how to attach vertices onto *FaceCircle* evenly. Vertex v_1 has the largest horizontal coordinate, so attach it to the rightmost point u_1 on *FaceCircle*. Then in counterclockwise order, v_2 to u_2, v_3 to u_3 and so on.

point on *FaceCircle*. Then starting from this vertex, counterclockwise attach all other vertices on the face to the corresponding target locations on *FaceCircle*.

5.3 Multiple target positions

In the real metro map, most of the vertices have two or more incident faces, which means that such vertices have to be attached to multiple different circles simultaneously. Figure 5.6a is an example of this case. We can find the target positions for a vertex on each *FaceCircle* using the method discussed in the previous section and obtain the displacement vectors for this vertex. Then the final location of v is simply the average of the sum of the vectors. Figure 5.6b is an example to explain this vector average method. After attaching all vertices to their target positions, we can move all the vertices simultaneously in one step.

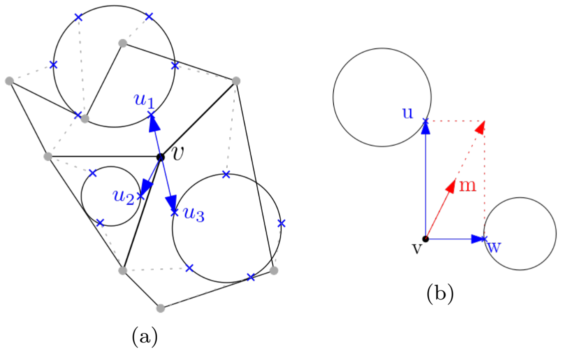


Figure 5.6: Examples to show the displacement vector. (a): An example to show vertex have to be attached to multiple different circles simultaneously. Vertex v has three incident faces, and on these three faces, it should be mapped to u_1, u_2 and u_3 in turn. The final location of v is $\dfrac{\overrightarrow{vu_1}+\overrightarrow{vu_2}+\overrightarrow{vu_3}}{3}$. (b): An example to show using the average of the displacement vectors as the target position. Vertex v has two target positions and the displacement vectors are $\vec{vu}$ and $\vec{vw}$. The final location of v is $\dfrac{\overrightarrow{vu}+\overrightarrow{vw}}{2}$.

5.4 Preserve the topology

No matter what layout style is used, maintaining topology is an essential criterion for metro map optimization. We have to make sure that there is no new intersections and the edge orderings remain unchanged during the optimization. One advantage of the Attach Circle approach is that we can update the coordinates of all vertices at the same time. However, updating the coordinates of all the vertices at the same time makes it difficult to maintain the topology. As is shown in Figure 5.7, we can find the target positions on each circle for all vertices and can know the final displacement vectors. However, we do not know whether the topology structure will be broken after moving all vertices. We need to check each edge for new intersections and check whether the edge ordering of each vertex has changed. This makes the algorithm very inefficient. Besides, if we detect that the topology is broken, we have to return all the vertices to their original position, so the metro map optimization fails.

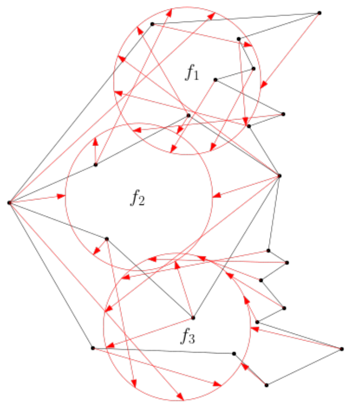


Figure 5.7: An example to show moving all vertices at the same time is difficult to preserve the topology. We can not know whether the topology structure will be broken after moving all vertices and do not know which vertex on which face are causing the problem.

An improved method is to divide the displacement vector evenly into many small parts and move one small part in each step. Check the topology once for each step. If the topology is broken, only this small step is reversed. Figure 5.8 is an example of this method. However, this method also has disadvantage. If one edge breaks the topology after only a few small steps, the rest of the edges and vertices will be stopped optimizing and the map will look almost identical to the original map. Moreover, because this method checks the topology once every small step, the algorithm is more inefficient.

We want to prevent the case that the entire face stops being optimized because one vertex destroys the topology. Instead of moving all the vertices at the same time, we move vertices one by one. For each vertex, divide its displacement vector evenly into small parts and move one small part in each step. In each step check whether all adjacent edges of this vertex will introduce new intersections or change the edge ordering of some vertices. If the topology is broken, reverse the

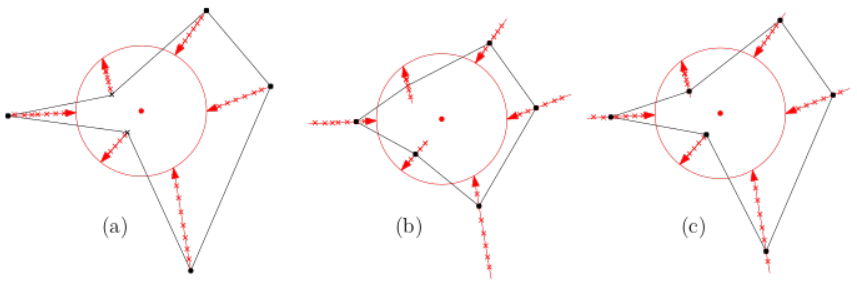


Figure 5.8: An example to show moving all vertices for a small step each time. (a): The input of a face. The displacement vector for each vertex is drawn in red. Every time move a small step at each vertex. (b): If the topology is broken, only one small step is reversed. (c): If one edge breaks the topology after only a few small steps, the rest of the edges and vertices will be stopped optimizing too.

step for the vertex and begin to move next vertex. Figure 5.9 is an example to show the process. The advantage of this method is that one vertex breaks the topology will not affect the continued optimization of other vertices. The disadvantage is that this method is still inefficient, because the topology has to be checked every step of each vertex move.

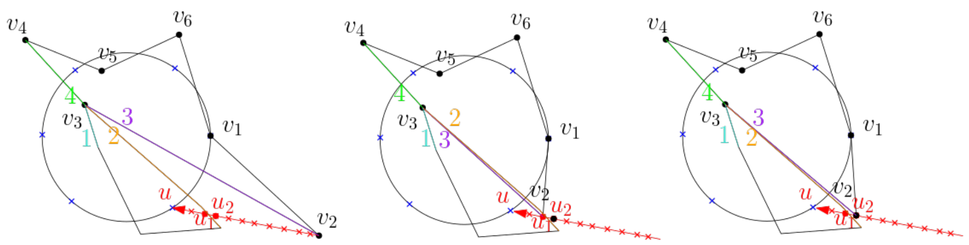


Figure 5.9: An example to show moving the vertices one by one while preserve the topology. The target position for vertex v_2 is u, but when v_2 move to u_1, the topology is broken. The edge ordering for vertex v_3 changes from 1234 to 1324. Thus, v_2 have to go back a step to u_2.

5.5 Results

We use the real metro map in Sydney to test the algorithm and we test three cases where the optimal edge length of the regular polygon was defined as 5, 10, 15. We calculated the average edge length, edge length variance, longest edge length, and shortest edge length before and after map optimization. For the area distribution, we calculate the difference between each face area and its optimal area. Then we calculate the average area difference, the variance of area difference, the maximum area difference and the minimum area difference before and after map optimization. The results for edge balancing are shown in Table 5.1 and the results for area distribution are shown in Table 5.2. The metro map results are shown in Figure 5.10.

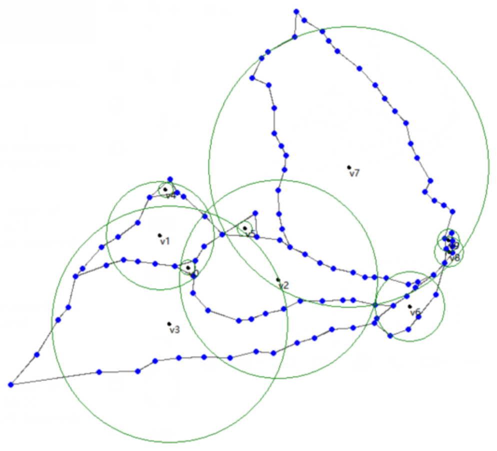
The edge balancing results				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	15.06	54.99	66.52	3.77
Edge length = 5	11.26	168.29	73.27	0.5
Edge length = 10	13.23	96.82	57.63	2.48
Edge length = 15	15.92	90.49	56.90	0.83

Table 5.1: The edge balancing results before and after the optimization.

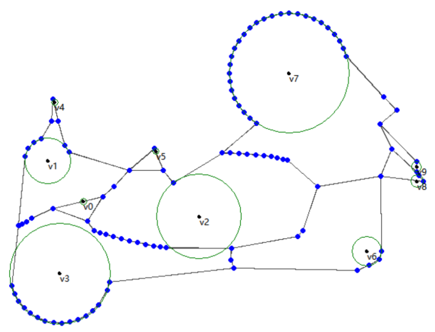
The area distribution results				
	Average difference	Area difference variance	Maximum	Minimum
Before(length=5)	3051	19775541	13501	3
After(length=5)	1265	1704891	3619	13
Before(length=10)	584	458855	1966	1.6
After(length=10)	546	424036	1635	11
Before(length=15)	4473	38348598	17258	10
After(length=15)	2487	12054969	9769	56

Table 5.2: The area distribution results before and after the optimization.

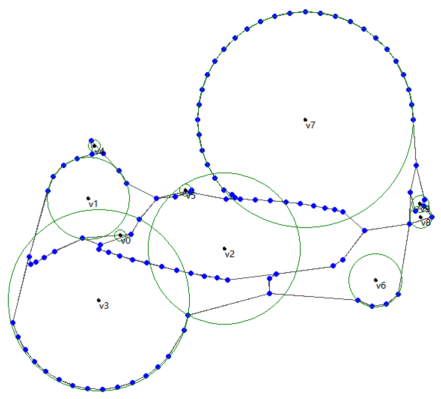
From Table 5.1 we can see that the the average edge length does not change much. But the variance of the edge length increases significantly, which indicates that the edge length of our metro map is more uneven after optimization. As can be seen from Figure 5.10, some vertices can



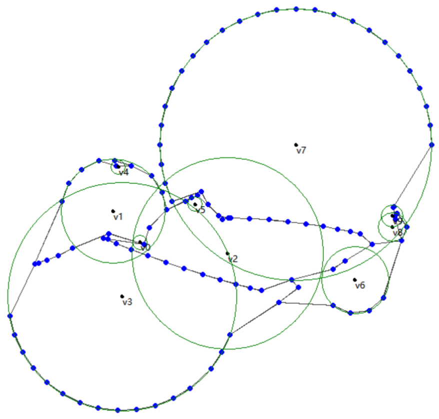
(a) Input



(b) Optimal edge length of the regular polygon is 5.



(c) Optimal edge length of the regular polygon is 10.



(d) Optimal edge length of the regular polygon is 15.

Figure 5.10: The results of the Sydney Metro map optimized by Attach Circle approach.

be perfectly attached to the circle with equal distance between them, but there are often extremely long edges at the junction of the two circles, resulting in a particularly unbalanced overall edge length of the metro map. And the smaller the optimal edge length of the regular polygon, the smaller the *FaceCircle*, the greater the imbalance. This is because the smaller the *FaceCircle*, the longer the edges are required at the connection of the circles, so the greater the difference in edge length from edges between the vertices attached to the circle. To be specific, the long edges are caused by poor distribution of the centroids. For example in Figure 5.10b, the centroid v_1 is too far away from centroid v_3, the common vertices of the two faces are therefore have to be distributed in the middle of the two *Facecircles*, far away from their target positions that are exactly on the *Facecircles*. Besides, there exists unexpected bends in the metro map, which make the shape of the faces very irregular.

Although the algorithm does not perform well in evening edge length, it gets good results in balancing area distribution. As is shown in Table 5.2, both the average area difference and the variance of the area difference are greatly reduced. This means each face on the metro map is closer to the optimal area than before optimization. A decrease in variance of the area difference means that the faces are more evenly distributed, which is exactly what we expect. Therefore, in the next approach, we can continue to use a similar method to improve the area distribution, but have to propose a new method to balance the edge length.

Chapter 6

Final Approach: Force-directed

This chapter discusses our ideas for the final approach. In this approach, we use a high level 'face-based' force-directed approach to obtain a better area distribution and we use a collapse-and-reinsert method for degree-2 vertices to balance the edge length. Then we deal with the dangling edges to make the result to be a real metro map. We give an introduction and the pseudo-code of this approach in Section 6.1. Then we describe why and how we rearrange the centroids to get a better area distribution in Section 6.2. In Section 6.3 we show how we use force-directed method to locate the degree-3+ vertices and In Section 6.4 we show how we reinsert the degree-2 vertices. In 6.5 we show the results of our algorithm. The dangling edges are processed in Section 6.6 and the final results and the evaluations are shown in Section 6.7.

6.1 Introduction

We learn from the second approach that poor centroid distribution results in unbalanced long edges and causes sharp bends. But despite these problems, the area distribution is better than the original metro map. The difference between the real area with the optimal area is greatly reduced and the variance of the area difference is smaller. The face with more vertices has a larger area, and is approximately proportional to the number of vertices on the face. In addition, the distance between the vertices that can be attached exactly on the circle is relatively uniform, which is also in line with our expectations. So, to sum up, in our final approach, we can use the same design criteria as Circle Attach approach but we have to come up with a different optimization approach to deal with sharp bends and long edges.

We propose a 'face-based' force-directed approach and the pseudo-code is shown in Algorithm 3. The idea is shown in Figure 6.1. We first rearrange the centroids of the faces, making the adjacent *FaceCircle* to be exactly tangent. Then we rotate the centroids, adjusting their common chain length between two degree-3+ vertices. Next the degree-3+ vertices are placed somewhere between the junction of all its incident faces using a force-directed method. Then we reinsert the degree-2 vertices on the chain between two degree-3+ vertices. The chain between two faces can reflect the

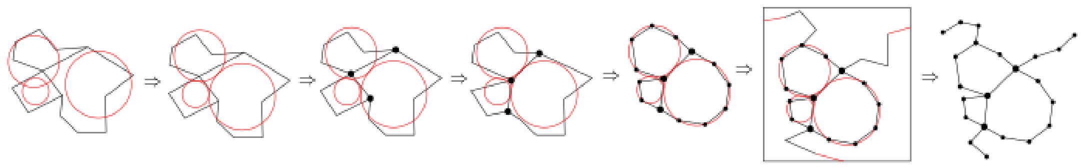


Figure 6.1: The optimization process of the final force-directed approach. First rearrange the centroids, then adjust the degree-3+ vertices. Then reinsert the degree-2 vertices onto the chains between the degree-3+ vertices. Then add a box to include dangling edges and remove the box at last.

force that is exerted by the centroids and the force is proportional to the size of the face. Finally we add a box around the metro map so that the dangling edges are included in the faces. After processed the dangling edges, the box will be removed.

The pseudo-code of the second optimization algorithm

Algorithm 3: Approach 3: Force-directed

```
Input: The routes of the metro map and the original geographic coordinates of all
       stations on the routes.
Output: The optimized station coordinates.
1 Add a box around the whole metro to deal with the dangling edges ;
2 for each face do
3     Calculate the centroid and FaceCircle as in the Grid Map approach;
4 Use a force-directed method to rearrange the centroids;
5 Rotate the centroids to adjust their common chain length;
6 Use a force-directed method to place the degree-3+ vertices;
7 for each chain between two degree-3+ vertices do
8     Reinsert the degree-2 vertices evenly along the chain;
9 Remove the box;
```

6.2 Rearrange the centroids

In the Circle Attach approach, poor centroid distribution results in long edges (see Figure 6.2a). The centroids of the two faces are too far apart, so their common vertices have long displacement vectors. The common vertices ends up somewhere near the middle of the two centroids, but far away from the optimal position on the circle. Thus, we decide to rearrange the centroids of the faces so that the distance between the two centroids is just appropriate for the displacement vector, as is shown in Figure 6.2b.

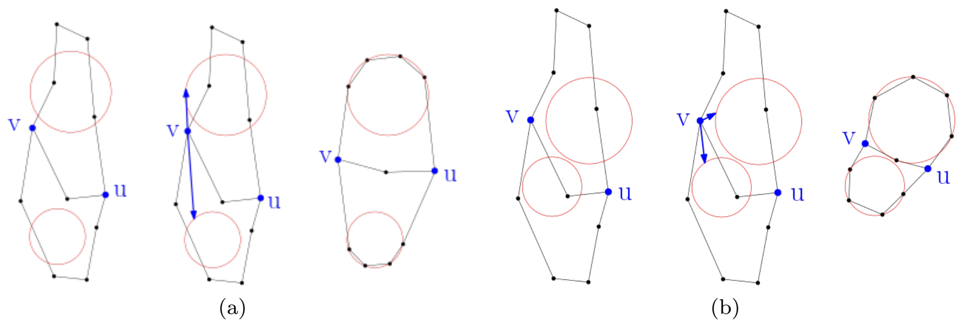


Figure 6.2: The idea of rearranging the centroids. (a): An example to show poor centroid distribution results in long edges. Vertex v and u have two incident faces and the centroids of the two faces are far away from each other. Thus the displacement vectors of v and u have very long length and the final location of v and u will be somewhere near the middle of the two $FaceCircles$, deviating far away from the target positions of both circles. (b): An example to show rearranging centroids can improve the long edge problem. The centroids of the two faces are moved closer to shorten the length of displacement vectors, and the result is much better with regard to the long edge problem.

We observe that for two adjacent faces, the farther apart their centroids are, the worse the long edge problem is. But the centroids should not be too close together either, because that would cause clusters of stations (see Figure 6.4). When two *FaceCircles* are perfectly tangent, there are most likely neither long edges nor clusters. Figure 6.3 give an example of this observation. Therefore, we define that the optimal distance between the centroids of two adjacent faces should be exactly the sum of the radius of the two *FaceCircles*. In other words, all faces that are adjacent should be tangent. Figure 6.4 demonstrates our idea about rearranging the centroids and our next step is optimizing the distance between the centroids to approximate the ideal distance.

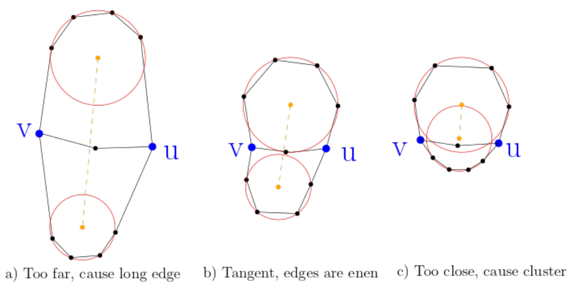

v
u
v
u
v
u
a) Too far, cause long edge
b) Tangent, edges are enen
c) Too close, cause cluster

Figure 6.3: An example to show our observation that when two *FaceCircles* are tangent, the distribution of vertices is most even. Too far of the distance leads to long edges, too close leads to clusters.

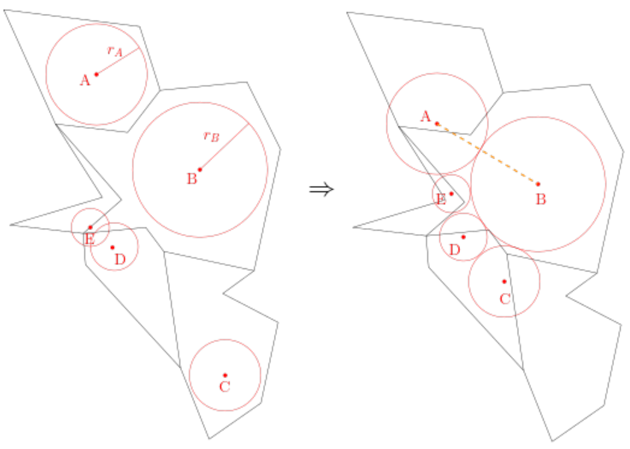


Figure 6.4: An example to show our idea about rearranging the centroids. The figure on the left is the original input of a metro map and the centroids and *FaceCircles* are drawn with red pen. The figure on the right shows the ideal distribution of centroids after rearranging. *FaceCircles* of adjacent faces are exactly tangent and the distance between two centroids are the sum of the radius of *FaceCircles*.

6.2.1 Distribute centroids

Since the process of rearranging the centroids does not involve stations and routes, we first ignore the vertices and edges of the metro map, and abstract the problem, solely considering the centroids graph (see Figure 6.5). We keep the *FaceCircles* in our model to facilitate the observation and connect the adjacent centroids with straight lines. The centroids are vertices V in the centroid graph model $G = (V, E)$ and the straight lines are edges E. Since we specify that the optimal distance between the two centroids is the sum of the radius of the *FaceCircles*, we can assign each edge of the centroid graph an ideal length. Therefore, our centroid graph model can be seen as a weighted graph, with the ideal length as the weight for every edge.

Inspired by the Force-directed graph drawing algorithm of Fruchterman et al. [18] that we introduced in the Preliminary Chapter (2.4), we decide to use a weighted force-directed model to optimize our weighted centroid graph. More specifically, we regard the centroids to be steel rings and the edges between them to be springs, and the ideal length of edges is like the springs in the natural state. Therefore, if the real distance is shorter or longer than the ideal length, the spring will be squeezed or stretched and thus give the centroids an attractive or repulsive force. Eventually the whole system will reach an equilibrium state where all real distances between adjacent centroids are equal to the optimal length. The example in Figure 6.6 shows the weighted force-directed model.

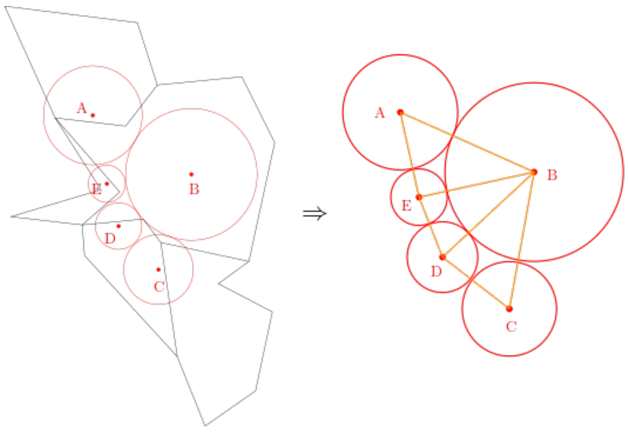


Figure 6.5: An example our centroid graph model. We only keep centroids and *FaceCircles* in our model to convenient the observation and connect the adjacent centroids with straight lines.

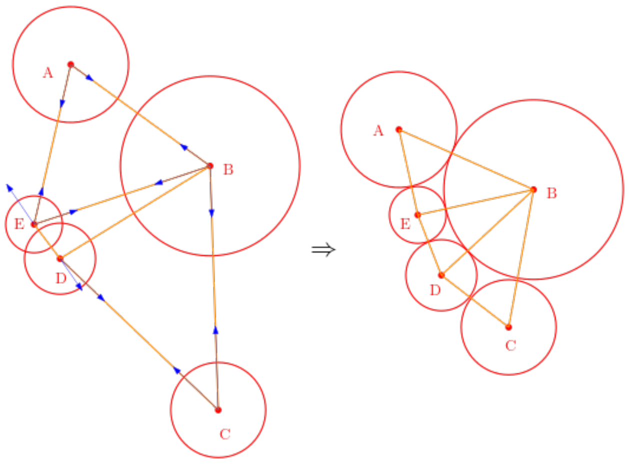


Figure 6.6: An example to show the weighted force-directed model. Edges are regarded as springs and will give attractive or repulsive force to vertices. Eventually the whole system will reach equilibrium and the adjacent circles will be approximately tangent.

Specific notes of our program

Although we use the same force-directed model as Fruchterman et al., we make some adaptations specific to our metro map optimization project. We list our adaptations below.

Force application

In both the force-directed model of Fruchterman et al. [18] and Eades [17], the forces are calculated in the following way: attractive forces are calculated only between neighbours but repulsive forces are calculated between every pair of vertices. However, this does not apply to our metro maps because many of the long routes to the suburbs need to be curved to magnify the concentrated downtown. Instead we apply forces in the following way: both attractive forces and repulsive forces are calculated only between neighbours. Figure 6.7 and Figure 6.8 show an example of the difference between our force application and Fruchterman et al. We believe our adaptation allows us to produce a more reasonable result for our metro maps.

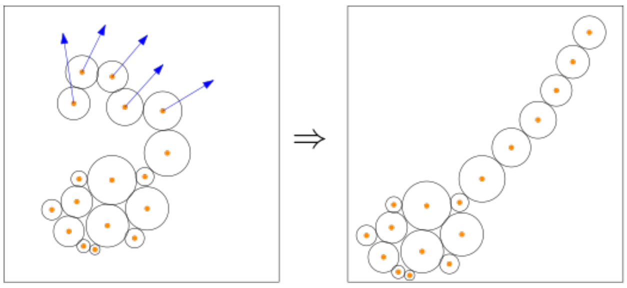


Figure 6.7: An example to show the result metro map using the force application of Fruchterman et al. Since non-adjacent faces also have repulsive forces, faces along the long route are repulsed into straight lines that extend to the upper right corner.

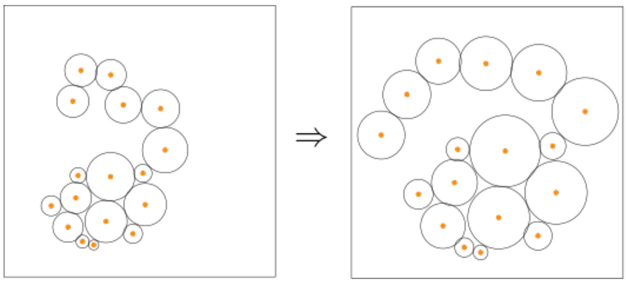


Figure 6.8: An example to show the result metro map using our force application. Because there are no repulsive forces between non-adjacent faces, the long routes can be bent into the empty space on the map, thus enlarging the map as a whole.

Minimum energy

In the real metro maps, sometimes adjacent faces can not be perfectly tangent, as in the example in Figure 6.9. Therefore, we can calculate the energy sum of the entire system to set the end

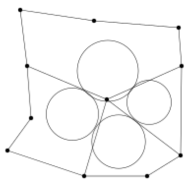


Figure 6.9: An example to show the case that adjacent faces cannot be tangent. The four faces are adjacent to each other, so the circles should be tangent to each other. But when the left and right circles are tangent, the upper and lower two circles cannot be tangent.

condition of the algorithm exactly the same as Fruchterman et al:

$$\sum (|p_i - p_j| - l)^2 \tag{6.1}$$

where $|p_i - p_j|$ is the real distance between two centroids and l is the ideal length between them. We minimize this formula.

6.2.2 Rotate centroids

In our experiments, we found that just making adjacent faces tangent still does not solve some of the clusters. For example in Figure 6.10, if there are many vertices on the common edges of two faces, then when the two centroids are moving closer and the vertices are attached to the *FaceCircles*, the vertices will become very crowded. Thus, we should increase the length of the common edges with many vertices.

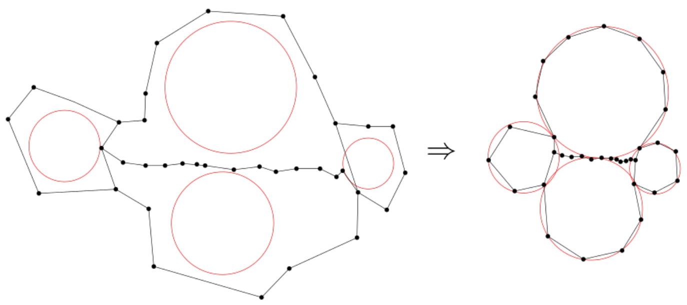


Figure 6.10: An example to show the cluster after two centroids are moved closer.

We increase the length of the common edges with many vertices by rotating the centroids of two adjacent faces, separating them. Our main idea is shown in Figure 6.11 and we use face A as an example. We want to distribute vertices evenly on the face and suppose the total number of vertices on face A is x. In the centroid graph, the centroid of face A has two adjacent edges with its two neighbors B and C. Suppose the vertices on face A that are contained between edges AB and BC is k. We define that k is half the number of vertices on the chains between face A and face B and C. To be specific, the chain between face A and face C is v_1v_2 and has y vertices, and the chain between face A and face B is v_2v_4 and has z vertices. The vertices contained between edge AC and AB should be $k = (y + z - 1)/2$. If we distribute the x vertices evenly on the *FaceCircle* of A, the angle between edge AB and AC should be $\frac{k}{x} * 2\pi$. Thus, we should rotate centroids C and B to change the angle $\angle CAB$.

The above rotation idea is only for a single face A and faces B and C are not adjacent in the example shown in Figure 6.11. However, when rotating the centroids of all the faces of the entire metro map, the vertices on face A cannot be perfectly evenly distributed on its *FaceCircle* as shown in Figure 6.11b. In the real metro map, faces B and C are very likely adjacent so that the centroids of B and C cannot rotate so much. Otherwise there will be long edges on faces B and C. Therefore, each centroid receives attractive and repulsive forces from distributing centroid process, as well as a rotational force from rotating centroid process. Figure 6.12 is an example to show the fores applying to the centroids. The centroids are subjected to a combination of forces and eventually the whole centroid graph reaches a dynamic equilibrium. Figure 6.13 is the result of the rearranging of the centroids of Sydney metro map. It is obvious that the distribution of centroids is better. The *FaceCircles* of adjacent faces are approximately tangent, while maintaining the topology of the centroid graph. Note that there is a possibility of destroying the topology during the rotation step, but in our practice, this rarely happen.

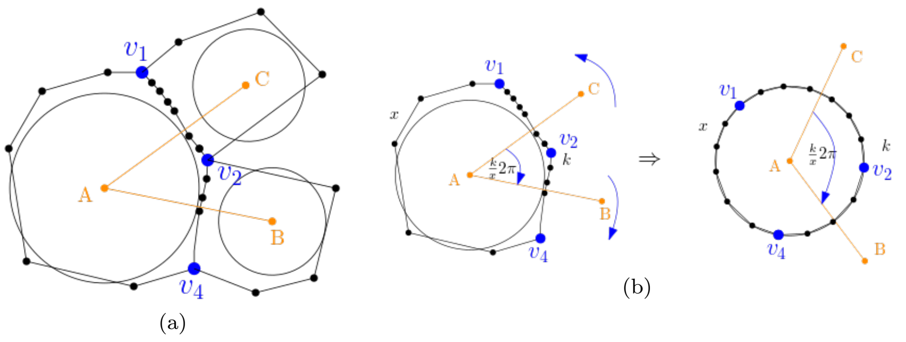


Figure 6.11: The idea of rotating the centroids. (a): There are 17 vertices on face A. The chain between face A and its neighbour face C is v_1v_2 and there are 8 vertices on chain; the chain between face A and its neighbour face B is v_2v_4 and there are 5 vertices on chain. Thus, we define that there should be $(8 + 5 - 1)/2 = 6$ vertices between edge AC and edge AB, and the angle between edge AC and edge AB should be $\frac{6}{17} * 2\pi$. (b): Rotate centroids B and C to change the angle.

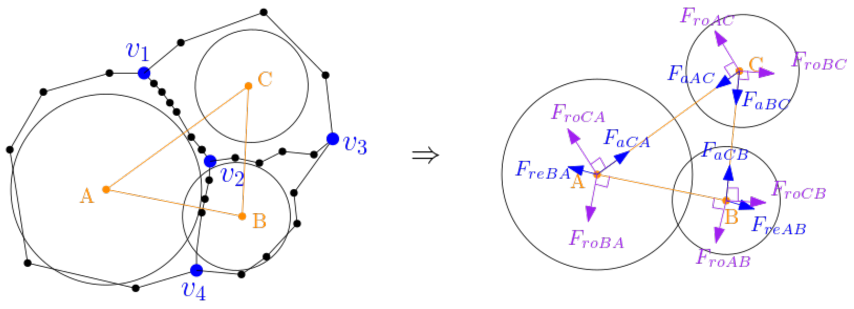


Figure 6.12: An example to show the fores applying to the centroids. The blue arrows represent the attractive forces and the repulsive forces from centroids distribution process. For example, F_{reBA} is the repulsive force to A from B and F_{aCA} is the attractive force to A from C. The purple arrows represent the rotation forces from centroids rotation process. For example, F_{roBA} is the rotation force to A from B and the rotation force is perpendicular to the edge AB.

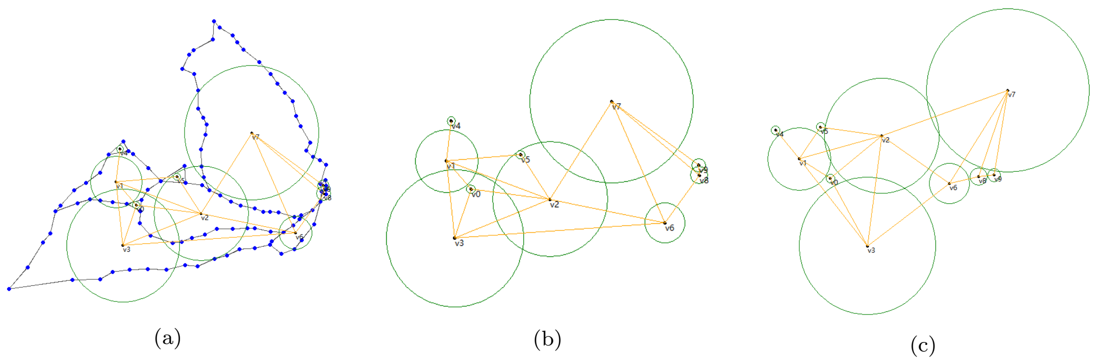


Figure 6.13: The result of the rearranging of the centroids of Sydney metro map. (a): The original input of Sydney metro map. (b): Extract the centroid graph model. (c): The new distribution of centroids after the rearranging process.

6.3 Placing the inside vertices

After obtaining a better distribution of the centroids, we have to place vertices and edges on the $FaceCircles$. Hong et al. [19] proposed a method to reduce the running time of their algorithm in their paper, called degree-2 collapse method. They first removed all degree-2 vertices to simplify the graph. Then their algorithm places the degree-3+ vertices at the right place. And then the degree-2 vertices are reinserted evenly on the chains between degree-3+ vertices. Following Hong et al. [19], we decide to use the same degree-2 collapse method to simplify our algorithm. When all the degree-2 vertices are removed, there are only degree-3+ vertices left in our metro map. We divide the degree-3+ vertices into two types, the inside degree-3+ vertices (which are not adjacent to the outside face) and the outside degree-3+ vertices (which are adjacent to the outside face). Accordingly, chains between the degree-3+ vertices are also divided into inside chains (not have the outside face as the incident face) and outside chains (have the outside face as the incident face). Figure 6.14 shows the difference between them. We will deal with the inside vertices and the outside vertices in different ways.

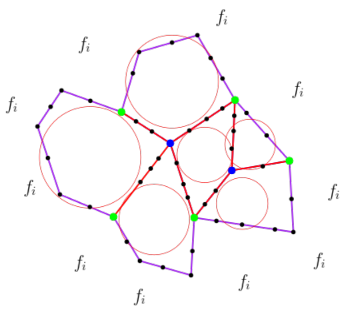


Figure 6.14: An example to show the difference between inside degree-3+ vertices and outside degree-3+ vertices. The outside degree-3+ vertices (green) are adjacent to the outside face but the inside degree-3+ vertices (blue) not. The outside chains (purple) have the outside face f_i as one of the incident face but the inside chains not (red).

6.3.1 Placing the inside degree-3+ vertices

An inside degree-3+ vertex is located at the junction of several faces. Considering that the area of the face should be proportional to the number of vertices on the face, we define that the distance between the inside degree-3+ vertex and its adjacent centroids should also be proportional to the number of vertices on the faces, as is shown in Figure 6.15. The $FaceCircle$ of a face with more vertices has a larger radius, so the inside degree-3+ vertex has to be further away from the centroid of the face to be able to attached to the circle.

Ideally, the inside degree-3+ vertex should be somewhere in the middle of all its adjacent $FaceCirles$. However, after rearranging the centroid, it is very likely that the position of this inside degree-3+ vertex deviates from these $FaceCirles$ farther away. Therefore, we first move the inside degree-3+ vertex to the centroid of the polygon formed by the adjacent centroids, like Figure 6.16 shows. Then again, we use force-directed method here to achieve the proportional effect. We define the optimal distance between this inside degree-3+ vertex and the centroid of each adjacent face as the radius of the $FaceCircle$ of the face. That is, connecting the inside degree-3+ vertex with the centroid, the intersection of the line and the circle is the optimal position of this vertex on the circle (blue dots in Figure 6.17). If the distance between the inside degree-3+ vertex and the centroid is shorter than the radius of the $FaceCircle$, the centroid will give the vertex a repulsive force to push it further away; and if the distance between the inside degree-3+ vertex and the centroid is longer than the radius of the $FaceCircle$, the centroid will give the vertex an attractive force to pull it towards to the centroid. Both the attractive forces and the repulsive forces are proportional to the distance from the optimal position. When the inside degree-3+

vertex is near the optimal position of all neighbor faces, the sum of the deviation from the optimal position of all the neighbors is likely minimized. Figure 6.17 is an example of this force-directed method for the inside degree-3+ vertex.

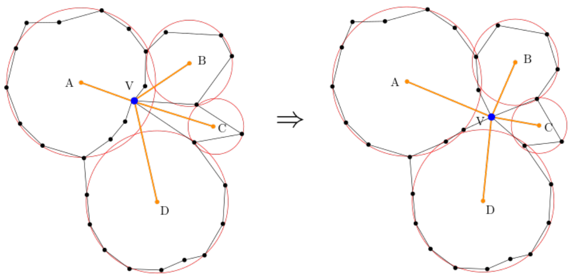


Figure 6.15: An example to show that the the distance between inside degree-3+ vertex V and the adjacent centroids are proportional to the number of vertices on the face. Face A has more vertices than face C, so the length of VA should be longer than the length of VC. Ideally we will have $\frac{VA}{r_A} = \frac{VB}{r_B} = \frac{VC}{r_C} = \frac{VD}{r_D}$.

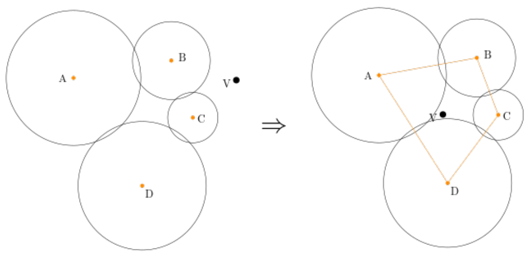


Figure 6.16: An example to show that we move the inside degree-3+ vertex to the centroid of the polygon formed by the adjacent centroids first.

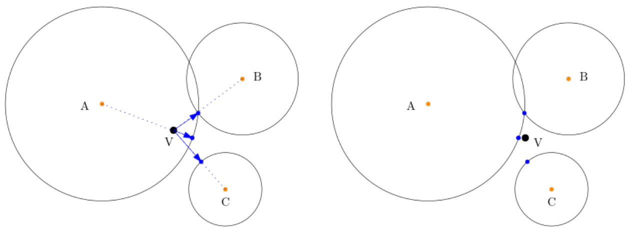


Figure 6.17: An example to show the force-directed method for the inside degree-3+ vertex. The distance between vertex V is shorter than the radius r_A, so centroid A has a repulsive force to V. The distance between vertex V is longer than the radius r_B, so centroid B has an attractive force to V. Vertex V deviates from the optimal position of centroid C more than B, so C has a bigger attractive force. Finally V fall near the optimal position of all neighbor faces.

6.3.2 Placing the inside degree-2 vertices

After placing the inside degree-3+ vertices, we reinsert the degree-2 vertices evenly along the chains between the degree-3+ vertices. The simplest method of reinserting is to draw a straight line between the two inside degree-3+ vertices and place the degree-2 vertices evenly on the line. Figure 6.18 is an example to shown this simple reinsert method. However, when there are two or more chains between the two inside degree-3+ vertices, this method makes the two chains overlap, as is shown in Figure 6.19. Instead, we use smooth arcs to represent the chains and reinsert the degree-2 vertices along the arcs.

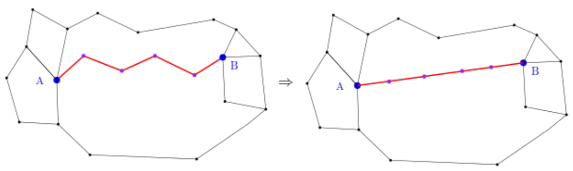


Figure 6.18: An example to show reinserting degree-2 vertices evenly on the straight line between two inside degree-3+ vertices.

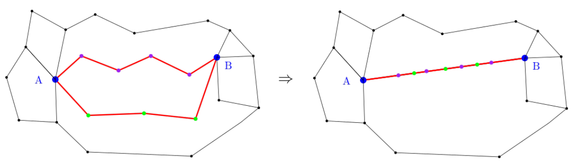


Figure 6.19: An example to show two or more chains between the two inside degree-3+ vertices will cause overlap.

For the inside degree-3+ vertices, we define that the distance between the vertex and the centroids of its adjacent faces are proportional to the number of vertices on the faces. Therefore, for the inside degree-2 vertices, we also have the same definition. To be specific, the inside chain has a proportional distance to its two incident faces. The more vertices on a face, the longer distance between the centroid and the chain. Considering that the points on the apollonian circle have a constant distance ratio to two fixed points, we decide to represent the chain by an arc of the apollonian circle. We regard the two centroids as two fixed points, and take the ratio of the two radius as the constant distance ratio. Then draw the apollonian circle of the two centroids. The distance from the points on the circle to the two centroids is the ratio of their radius and we reinsert the degree-2 vertices evenly along the arc of the apollonian circle. Figure 6.20 is the idea of this method.

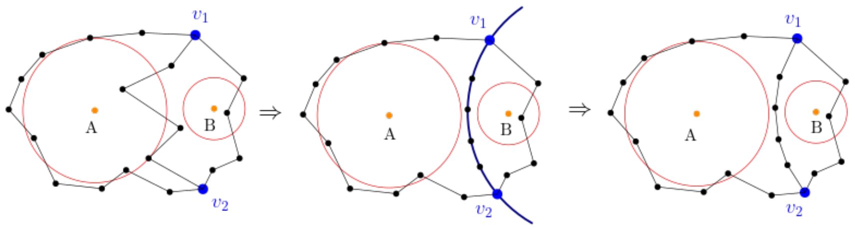


Figure 6.20: An example to show inserting the inside degree-2 vertices on the apollonian circle of the two centroids. Vertices v_1 and v_2 are two inside degree-3+ vertices and the chain between them has two incident faces A and B. We take the two centroids A and B as two fixed points and draw the apollonian circle of them (blue arc). Then we reinsert the degree-2 vertices of the chain on the arc of the apollonian circle evenly.

However, the example in Figure 6.20 is a special case where the apollonian circle A_{apo} passes exactly the two degree-3+ vertices. In most cases, the two degree-3+ vertices are some distance from the circle, so we need another two arcs to connect the two degree-3+ vertices with the arc of A_{apo}, as is shown in Figure 6.21.

We have to make sure that the two arcs go through the two degree-3+ vertices and connect with A_{apo} smoothly. We observe that the arcs that are tangent to A_{apo} can connect smoothly,

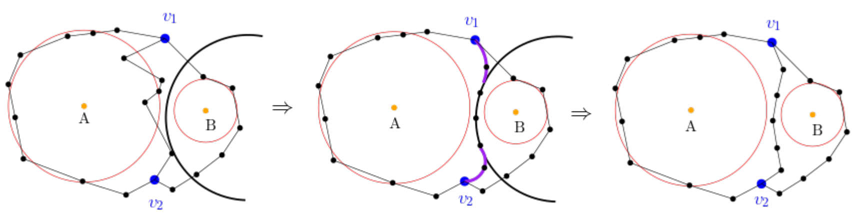


Figure 6.21: An example to show the apollonian circle A_{apo} does not go through the two degree-3+ vertices. We want to reinsert the degree-2 vertices on the arc of A_{apo}, so we need two more arcs (purple arcs) to connect the two degree-3+ vertices with A_{apo} smoothly.

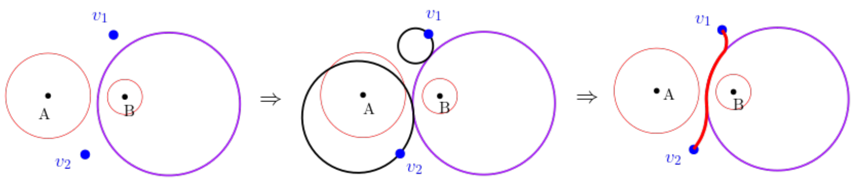


Figure 6.22: An example to show that we use circles that go through the two degree-3+ vertices and is tangent to A_{apo} to represent the two arcs. The purple circle is the apollonian circle for face A and B. The blacks circles go through the two degree-3+ vertices v_1 and v_2 and are tangent to the purple circle. Then we get a smooth three-part-arc (fat red line) between the two degree-3+ vertices.

as is shown in Figure 6.22. Thus, we decide to represent the two arcs with two circles that go through the two degree-3+ vertices and that are tangent to A_{apo}.

Because the vertices on the apollonian circle arc are proportional to the two centroids, we want the vertices on the chain to fall on the apollonian circle arc as much as possible, rather than on our two new connection arcs. Thus, we need to make the two new connection arcs as short as possible, that is, the smaller the radius of the connection circles, the better. Figure 6.23 is an example. However, we observe that when the radius is too small, although the apollonian circle arc is long, the two connection arcs are too curved, resulting in an unsmooth chain. Therefore, we have to make a trade-off between the apollonian circle arc length and the smoothness of the chain.

We define the radius for each connection circle as the distance between the degree-3+ vertex and the apollonian circle (see Figure 6.24 and Figure 6.25). Such a radius not only ensures that the connection circle passes through the degree-3+ vertex and is tangent to the apollonian circle, but it also does not bend the chain too much and does not make the apollonian circle arc too short.

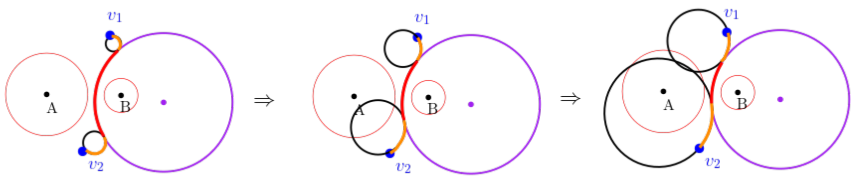


Figure 6.23: An example to show the effects of different radius for the two connection circles. When the radius is small, we can put more vertices on the apollonian circle arc (red arc), but the chain is too curvey. When the radius is too big, the chain is smooth but we can only put a few vertices on the apollonian circle arc.

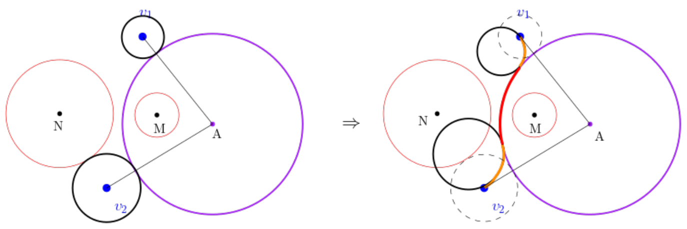


Figure 6.24: An example to show the radius for the connection circle when the degree-3+ vertex is outside the apollonian circle. The distance from degree-3+ vertex v_2 to the apollonian circle is $|Av_2| - r_A$ and this is the radius for the connection circle for vertex v_2.

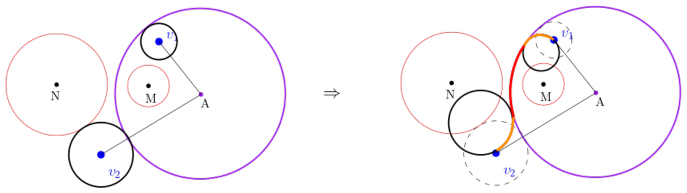


Figure 6.25: An example to show the radius for each connection circle when the degree-3+ vertex is inside the apollonian circle. The distance from degree-3+ vertex v_1 to the apollonian circle is $r_A - |Av_1|$ and this is the radius for the connection circle for vertex v_1.

6.4 Placing the outside vertices

After locating all the inside vertices, we start to deal with the outside vertices. Outside vertices have the outside face as an incident face and we can not calculate the centroid and *FaceCircle* for the outside face, so we have to propose a different optimization method for both outside degree-3+ vertices and degree-2 vertices. As with the inside vertices, we first determine the location of the outside degree-3+ vertices and then reinsert the degree-2 vertices into the chain between the degree-3+ vertices.

6.4.1 Placing the outside degree-3+ vertices

At first, we want to deal with the outside degree-3+ vertices in the same way we deal with the inside degree-3+ vertices. To be specific, we first put an outside degree-3+ vertex to the centroid of the polygon that formed by its adjacent centroids, then we use a force-directed method to make the distance to the cenroids proportional to the number of vertices. However, when we look at the real metro map, we find that there are two problems with dealing with outside degree-3+ vertices in this way. The first problem is that when both ends of a chain are outside degree-3+ vertices and they only have two incident faces, the chain will shrink into a single point, causing overlap. Because the two incident faces of the two outside degree-3+ vertices are the same, these two vertices will be placed in the same position on the line segment formed by the centroids of the two faces. Figure 6.26b is an example to show this problem. The second problem is that moving a outside degree-3+ vertex to the centroid of the polygon formed by the centroids of its adjacent faces may cause clusters. Figure 6.26c is an example to show this problem.

We observe that most outside degree-3+ vertices have only two inside incident faces (not include the outside face), so we first discuss the case of the outside degree-3+ vertices with only two incident faces. To solve this issue, we draw the apollonian circle of the two centroids, then the apollonian circle and the line segment will have an intersection. We call this intersection *proportional point*. Instead of putting the outside degree-3+ vertex directly on this intersection, we move the outside degree-3+ vertex along the apollonian circle to a certain distance from this proportional point, as is shown in Figure 6.27. Thus, our next step is to determine the distance from the intersection.

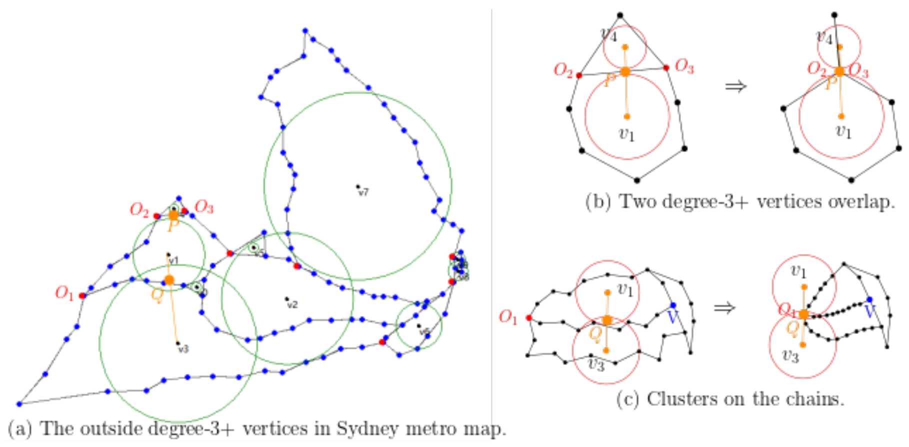

O_2
O_3
P
O_1
Q
v1
v2
v3
v7
(a) The outside degree-3+ vertices in Sydney metro map.
v_4
v_1
⇒
(b) Two degree-3+ vertices overlap.
V
v_3
(c) Clusters on the chains.

Figure 6.26: An example to show that if we use the same method as the inside degree-3+ vertices to deal with the outside degree-3+ vertices, there will have two problems. (a): The outside degree-3+ vertices of Sydney metro map are marked with red dots. (b): An example to show the first problem. O_1, O_2 and O_3 are three outside degree-3+ vertices. O_2 and O_3 are the endpoints of a same chain. The centroids of their incident faces are v_1 and v_4 and form a line segment. The centroid P of this segment is on the segment and force-directed method can only move A and O_2 along the segment and two vertices will be put at the same point. (c): An example to show the second problem. O_1 has two incident faces v_1 and v_3, and the centroid of the segment formed by the two incident centroids is Q. Then we use force-directed method to move Q, making it proportional to v_1 and v_3. Vertex V is an inside degree-3+ vertex and its location has already been determined. When dragging vertex O_1 to Q, the chain between O_1V will be too short, causing clusters.

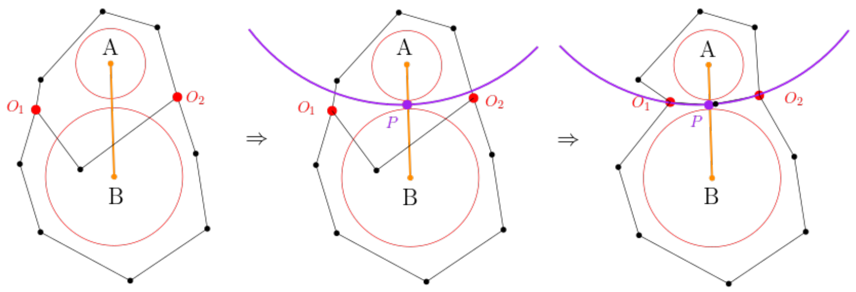


Figure 6.27: An example to show the idea of dealing with outside degree-3+ vertices. We draw the apollonian circle of centroid A and B, and the apollonian circle intersects line segment AB at P. P is called proportional point. We put the outside degree-3+ vertices on the apollonian circle, some distance from the proportional point.

To determine the distance from the proportional point, we draw a circle with the proportional point as the center, called the *positioning circle*, as is shown in Figure 6.28. The positioning circle and the apollonian circle will have two intersections, and we place the outside degree-3+ vertices at the two intersections. The problem is converted to determine the radius of the positioning circle.

We observe that the larger the radius of the positioning circle, the longer the chain between the two degree-3+ vertices. Too large the radius causes long edges on the chain, and too small the radius causes clusters on the chain, as is shown in Figure 6.29. Therefore, the radius should be proportional to the number of vertices on the chain. We define that the radius of the positioning circle is

$$r_{positioning} = \frac{(n-1)*l}{2} \tag{6.2}$$

where n is the number of vertices on the chain and l is the edge length of the regular polygon we input. Thus, the more vertices on the chain, the further away the outside degree-3+ vertex is from the proportional point and the longer the chain. This can avoid the long edges and clusters on the chain.

For outside degree-3+ vertices that have more than two incident faces, we do the above processing for all its incident faces with common edges. Finally take the centroid of the polygon composed of the resulting intersections as the position of the outside degree-3+ vertex. Figure 6.30 is an example of this case.

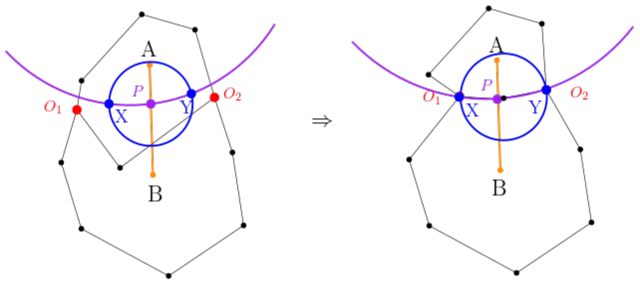


Figure 6.28: An example to show the positioning circle used to determine the locations of outside degree-3+ vertices. The apollonian circle of A and B (purple) intersects segment AB at the proportional point P. Take P as the center, draw a circle (blue). We call it the positioning circle. The positioning circle intersects the apollonian circle at two points X and Y (blue) and these two points are the positions for the outside degree-3+ vertices.

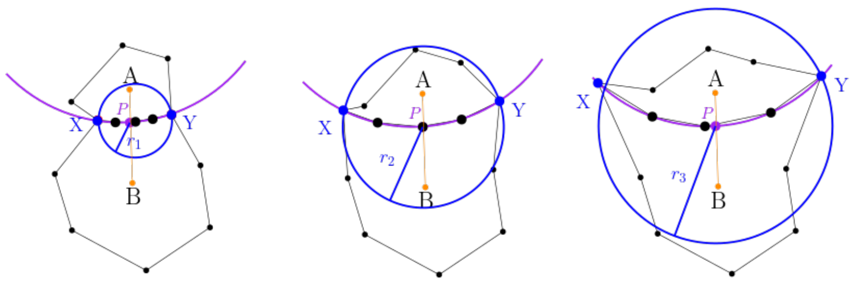


Figure 6.29: An example to show the radius of the positioning circle should be proportional to the number of vertices on the chain. Too large the radius causes long edges on the chain, and too small the radius causes clusters on the chain.

6.4.2 Placing the outside degree-2 vertices

We reinsert the outside degree-2 vertices in a way similar to the inside degree-2 vertices. The only difference is that instead of reinserting the degree-2 vertices on the apollonian circle arc, we reinsert them on the arc of the $FaceCircle$, as is shown in Figure 6.31. We choose the $FaceCircle$ because the vertices on the $FaceCircle$ are considered to have the optimal distance from the centroid. Similarly, we need another two arcs to connect the outside degree-3 vertices with the $FaceCircle$ smoothly. We define the radius for the connection circle is the distance between the outside degree-3+ vertex and the $FaceCircle$, as is shown in Figure 6.32.

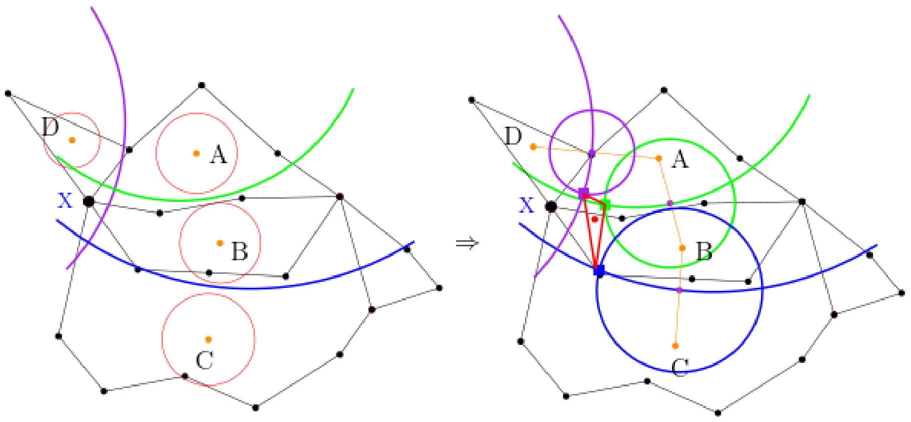


Figure 6.30: An example to show the outside degree-3+ vertex that have more than two incident faces. Vertex X is an outside degree-3+ vertex and it has four incident faces A, B, C and D. The apollonian circle and positioning circle for A and D is drawn in purple and their intersection is marked by a purple square. Similarly, for A and B the intersection is marked by a green square and for B and C is marked by a blue square. Then the three squares form a polygon and its centroid (red dot) is the final position for the outside degree-3+ vertex X.

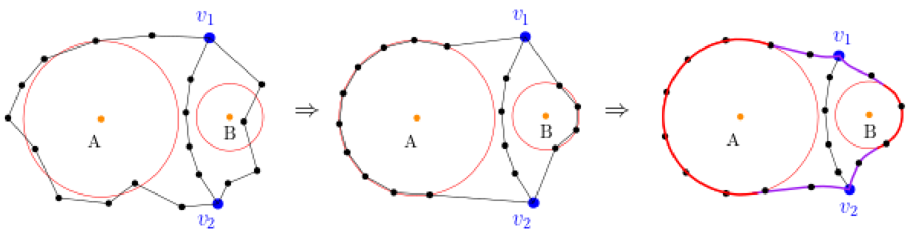


Figure 6.31: An example to show the outside degree-2 vertices are inserted on the *FaceCircle* arc (red arc) evenly. The purple arcs are the two arcs that can connect the outside degree-3 vertices with the *FaceCircle* smoothly.

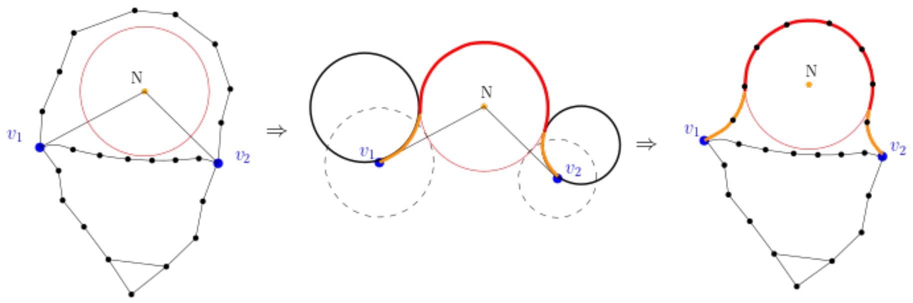


Figure 6.32: An example to show the outside three-part-arcs. Outside chain between vertices v_1v_2 is placed evenly on the *FaceCircle* (red arc) of face N and two connection arcs (orange arcs).

6.5 Results

We use the metro map of Sydney to test our approach and the input edge length of the regular polygon is 10. The result is shown in Figure 6.33. We first calculate the centroids and *FaceCirles* for each face. Then we rearrange the centroids with the force-directed method combined with a rotation of the controids. Next step we locate the inside degree-3+ vertices and the outside degree-3+ vertices. Finally we reinsert the inside degree-2 vertices and the outside degree-2 vertices evenly on the chains between degree-3+ vertices. Figure 6.33a is the original input and the centroids and *FaceCircles* are drawn. Figure 6.33b is the result after rearranging the centroids. Figure 6.33c is locating the degree-3+ vertices. The inside degree-3+ vertices are marked by red dots and the outside degree-3+ vertices are marked by purple dots. For ease of observation, we use the red lines to represent the inside chains, and the purple lines to represent the outside chains. Figure 6.33d is the final result of Sydney metro map and the degree-2 vertices are reinserted on the three-part-arcs.

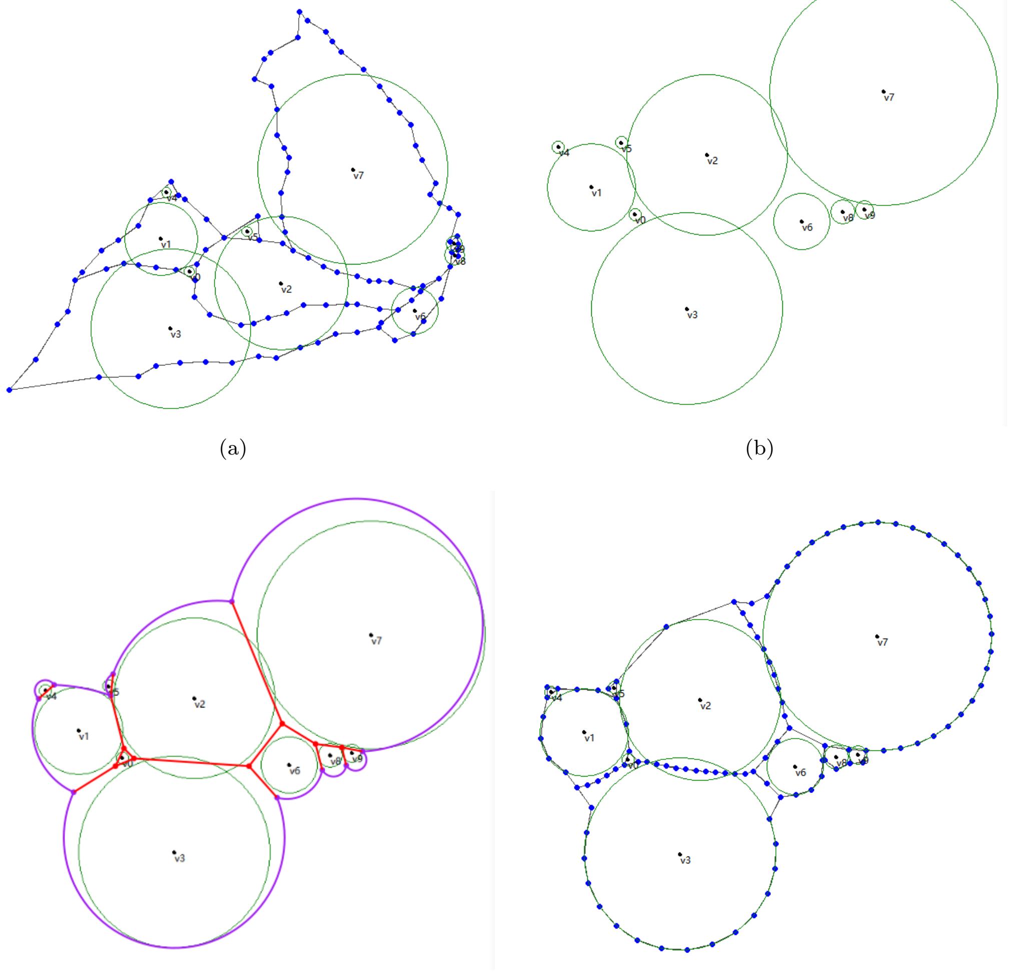


Figure 6.33: The result of Sydney metro map using our final approach. (a): Original input. (b): The result after rearranging the centroids. (c): Locating the degree-3+ vertices. The inside chains and outside chains are what we expect and are draw by hand. (d): The final result of our algorithm.

We believe that the final result is more readable than the original input. First, in the original metro, there exists sharp bends, making eyes difficult to track routes. In the final result, the routes are smoothly connected, and no sharp bends will disrupt the eye tracking. Second, the two faces on the right have clusters and stations overlap with each other, making it difficult to distinguish different stations. In the final result, stations and routes are separated by a certain distance, making it easier to distinguish stations. In addition, the faces have a more regular shape and size than the original map. Faces are more closer to the circle, leaving more space for the labels and separating two close routes apart. Last, the length of the edges is more uniform, and the distance between vertices on a chain is equal. But the distances between vertices on the different chains are not equal. Thus in some chains the points are relatively clustered, in others they are relatively sparse. In the later Section, we will give a more detailed evaluation and will give the data comparison.

6.6 Deal with the dangling edges

The results we have got so far are not really metro maps, because the stations on the dangling edges are not included. So our next step is to reinsert the dangling edges. In this thesis we are using a 'face-based' approach and we do not include outside face in our processing. Therefore, the dangling edges can only be processed if they are contained in an inside face. To include the dangling edges in a inside face, we first add a box around the entire metro map, and then manually add a line to connect the end vertex of the dangling edges to the border of the box, as is shown in Figure 6.34b. The connection points on the border are arbitrary, but we ensure they do not break the topology. This way all the dangling edges are contained in the inside faces and we can use our algorithm to optimize the whole metro map. Then after the optimization step, we remove the box and the connection lines (see Figure 6.34c). Then the result will be a real metro map with all stations and edges in.

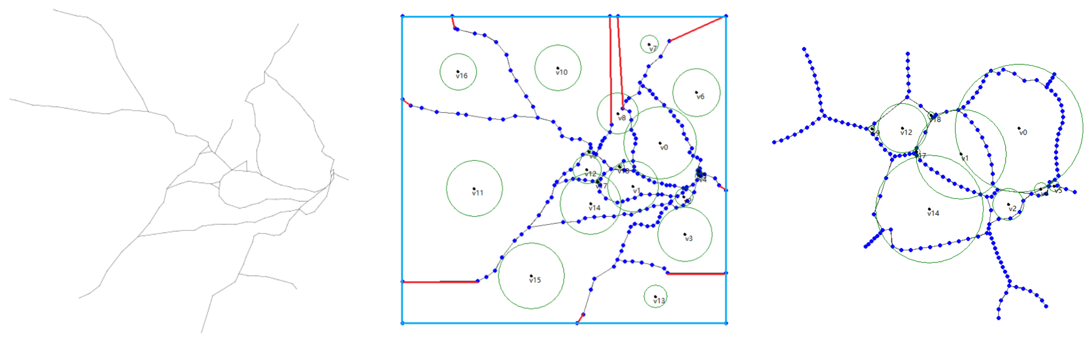

(a) (b) (c)

Figure 6.34: The idea of containing dangling edges in the inside faces by adding a box. (a): The original input of Sydney metro map. (b): We add a box (blue lines) around the metro map and connect the end vertex of the dangling edges to the border of the box (red lines). (c): After the optimization step, we remove the box and the lines drawn by hand.

6.7 Evaluation

Now we can optimize the entire metro map using our final approach. In this section, we test our approach with metro maps of five cities to check whether the algorithm can optimize all of them. The five cities we choose are Sydney, Wien, Montreal, Washington and karlsruhe. We show the optimization results of each metro map and analyze the data. For each metro map, we select three different values as the edge length of the regular polygon to see if the metro map can be optimized under each edge length. We compare the average edge length, edge length variance, maximum edge length and minimum edge length before and after algorithm optimization. This shows how well our algorithm works when balancing the edges length. In addition, we calculate the area difference between the faces and their optimal area before and after the optimization of the algorithm. We calculate the average area difference, the variance of the area difference, the maximum area difference and the minimum area difference. Besides, we calculate the average roundness of all the faces of the metro map to see if the shape of the faces are more regular.

6.7.1 Sydney

For Sydney metro map, the three different edge length of our input for the regular polygon are 2, 6, and 9 respectively. The results are shown in Figure 6.35, Figure 6.36, and Figure 6.37 respectively. From the figures, we believe that the readability of the metro map has been improved. The distance between stations is approximately equal compared to the original input. The original map routes are curved and have sharp bends, but the optimized routes are relatively smooth. The original map has close routes and clusters, but our results reduce the clusters and the faces look rounder. However, we can see that when the optimal edge length is 2, the topology is broken. We think this is caused by the three-part-arcs. Figure 6.38 is an example. When the radius of the apollonian circle and the two conection circles are too small, the arcs that used to reinsert degree-2 vertices will be too curve, resulting the intersection with other arcs. Besides, for each edge length, the faces that are adjacent to the outside face do not reach the maximum roundness and the shape

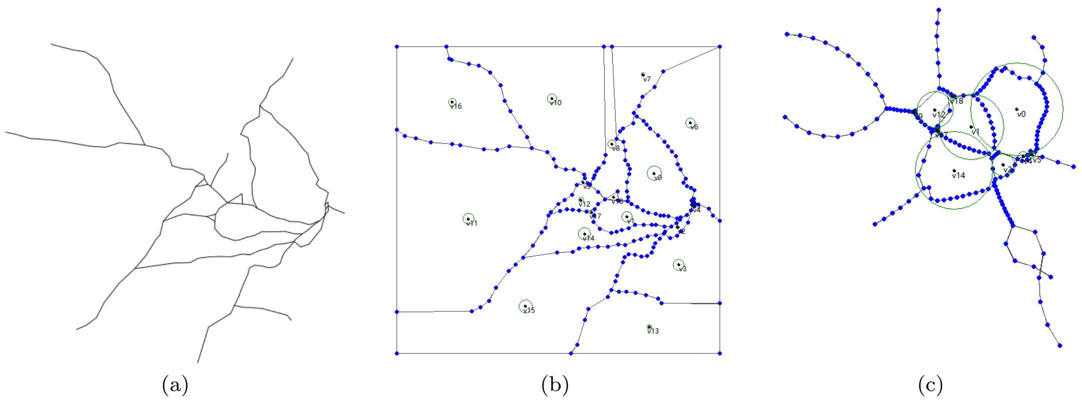

(a) (b) (c)

Figure 6.35: The result of Sydney metro map (optimal edge length = 2).

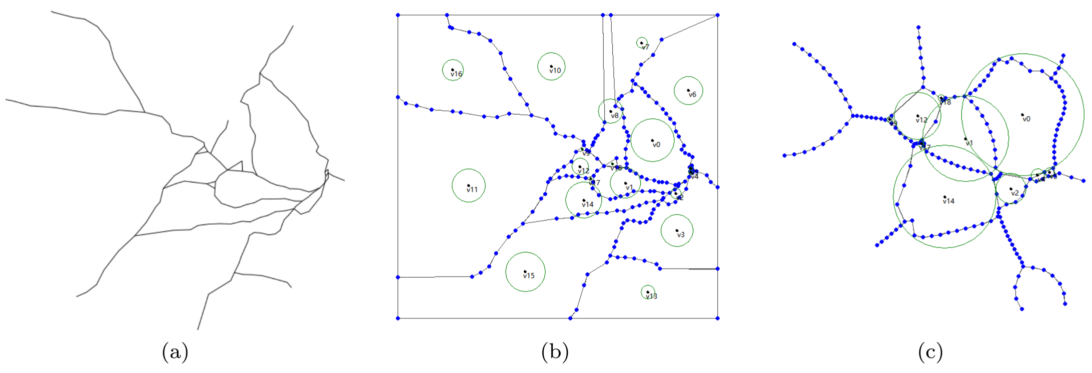

(a) (b) (c)

Figure 6.36: The result of Sydney metro map (optimal edge length = 6).

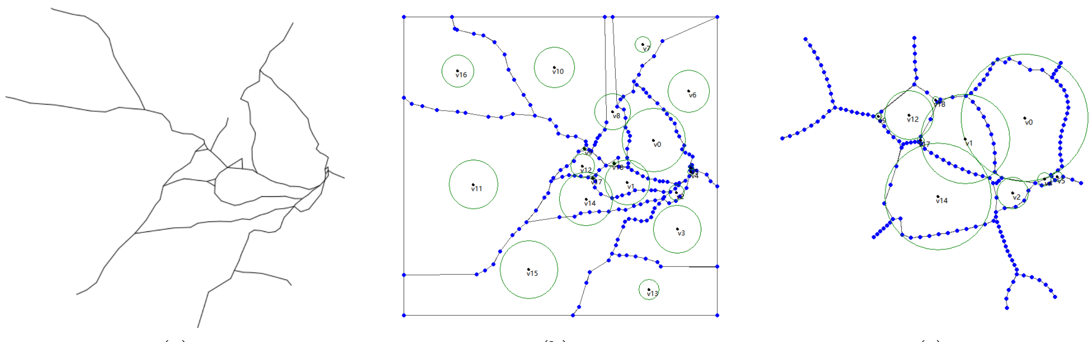

(a) (b) (c)

Figure 6.37: The result of Sydney metro map (optimal edge length = 9).

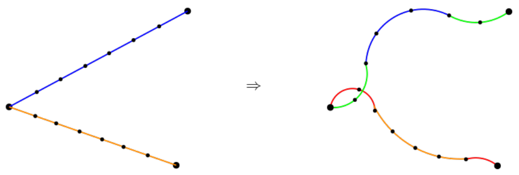


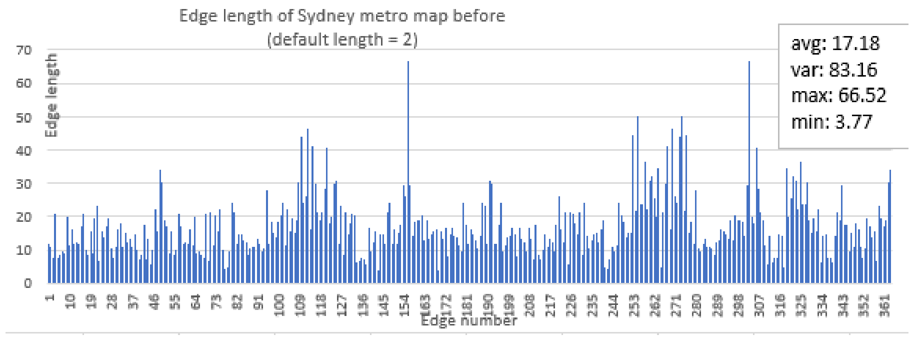
Figure 6.38: An example to show the intersection of two chains.

is not as regular as we expect. This is because these faces are pushed by the faces formed by the dangling edges and the box. We removed the added boxes and edges after the optimization was complete, but the added faces were always present during the algorithm optimization process.

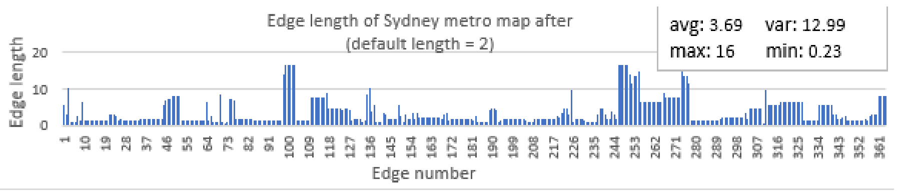
Next we use data to give a more detailed analysis. We draw the edge length before and after optimization into a bar chart, and the changes of the edge length are shown in Figure 6.39. The data of the three edge length before and after the optimization are collated into Table 6.1. For each optimal edge length, our algorithm can optimize the metro map, making the edge length more balanced. When we define that the optimal edge length is 2, the final average edge length of the whole metro map is shortened from 17.18 to 3.69. Similarly, when we set the optimal edge lengths to 6 and 9, the final average edge lengths are 5.71 and 8.04 respectively, both close to the optimal edge length. This means that our algorithm successfully optimizes the edge length, and the average edge length is close to the optimal edge length we input. Besides, for each optimal edge length, the variance of the edge length is greatly reduced. This indicates that the edge length of the metro map is more uniform and the distance between the two adjacent stations is approximately equal. A smaller variance of the edge length means that there are fewer long edges and clusters in the metro map. When we set the optimal edge length to be 6, the algorithm reduces the variance of the edge length from 83.16 to 10.11. This is better than when the edge length is 2 (from 83.16 to 12.99) or 9 (from 83.16 to 20.94). We believe that there is an optimal edge length setting (around 6 in Sydney metro map), which can achieve the maximum balance of the edge length of our metro maps.

The edge balancing results for Sydney metro map				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	17.18	83.16	66.52	3.77
Edge length = 2	3.69	12.99	16	0.23
Edge length = 6	5.71	10.11	29.12	1.31
Edge length = 9	8.04	20.94	42.40	2.01

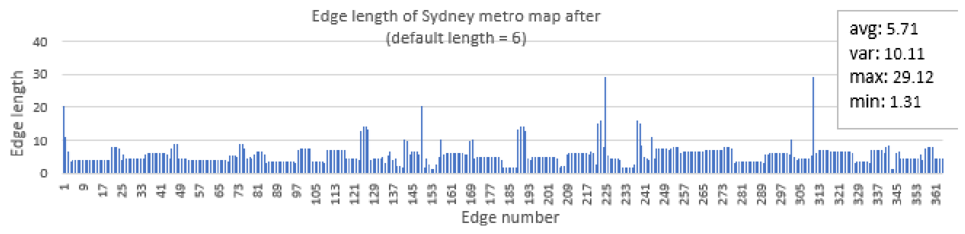
Table 6.1: The edge balancing results before and after the optimization for Sydney metro map.



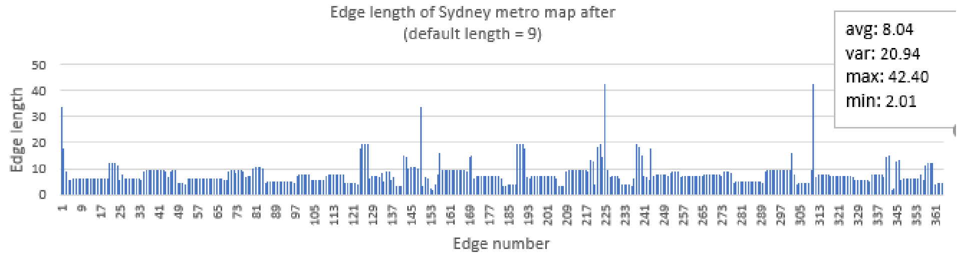
(a) Edge length of Sydney metro map before the optimization.



(b) Edge length of Sydney metro map after the optimization (optimal edge length = 2).



(c) Edge length of Sydney metro map after the optimization (optimal edge length = 6).



(d) Edge length of Sydney metro map after the optimization (optimal edge length = 9).

Figure 6.39: The result of edge balancing. The average edge length is reduced, which is more closer to our optimal edge length. The variance of the edge length also reduces, which means that edges are more balanced. The length of the edges of a chain are approximately the same, so there are many consecutive edges of the same length on a bar chart.

The data of the area difference before and after optimization are collated into Table 6.2. Different optimal edge lengths affect the size of the *FaceCircles* and influence the centroids layout after rearranging. For the area distribution results, the average difference of each edge length is reduced. This means the area of the faces on the metro map are closer to the optimal area. When the edge length is 2, the average difference between the real area and the optimal area is reduced from 3839 to 44. This means that almost all faces are close to their optimal area, and the area of a face is proportional to the number of vertices. The variance of the area difference are also reduced significantly. However, the roundness is reduced from 0.518 to 0.414 and we believe this is caused by the intersection. An intersection leads to an irregular shape of the faces, thus lowering the average roundness of the metro map. When the edge length are 6 and 9, both the area difference and the area difference variance are also reduced. The average roundness of the faces are increased, which means the shapes of the faces on the map are more regular and closer to the circle. This is exactly what we expect: to separate two close routes and increase the spaces for labels.

The area distribution results for Sydney metro map					
	Average difference	Variance	Maximum	Minimum	Average roundness
Before(edge=2)	3839	30716336	16731	31	0.518
After(edge=2)	44	5116	221	0.52	0.414
Before(edge=6)	2643	15006953	11810	0.61	0.518
After(edge=6)	390	391035	1909	4.79	0.627
Before(edge=9)	996	2517630	4889	46	0.518
After(edge=9)	831	1695975	4120	0.79	0.612

Table 6.2: The area distribution results before and after the optimization for Sydney metro map.

6.7.2 Washington

The three different edge length we choose for Washington metro map are 5, 10, and 15 respectively. The results are shown in Figure 6.40, Figure 6.41, and Figure 6.42 respectively. All the data are collated into Table 6.3 and Table 6.4. The analysis of Washington metro map is similar to the Sydney metro map. The average edge length is closer to the optimal edge length and the edge length variance is reduced. Especially when the optimal edge length is equal to 10, the edge length variance reduced from 151.03 to 13.89. We believe that our algorithm can significantly improve the edge length balance of Washington metro map and the optimal edge length should be around 10. The algorithm also improves the area distribution of Washington metro map. Both the average area difference and the variance of area difference are reduced. When the optimal edge length is 5, the average area difference is reduced from 1444 to 23 and the variance of area difference is reduced from 2538525 to 736. This means that almost all the faces are close to their optimal area. In other words, the area of a face is proportional to the number of vertices on the face. The roundness is increased so that the faces can leave more space for labels. The bar chart for the edge length before and after the optimization are shown in Figure 6.43.

The edge balancing results for Washington metro map				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	23.78	151.03	96.44	5.26
Edge length = 5	14.10	94.73	42.43	3.15
Edge length = 10	13.23	13.89	39.20	6.14
Edge length = 15	16.02	25.18	60.21	9.29

Table 6.3: The edge balancing results before and after the optimization for Washington metro map.

The area distribution results for Washington metro map					
	Average difference	Variance	Maximum	Minimum	Average roundness
Before(edge=5)	1444	2538525	4032	30	0.64
After(edge=5)	23	736	70	3	0.73
Before(edge=10)	1034	1681262	3192	2	0.64
After(edge=10)	126	9937	262	25	0.71
Before(edge=15)	548	525127	1793	56	0.64
After(edge=15)	281	51052	603	59	0.69

Table 6.4: The area distribution results before and after the optimization for Washington metro map.

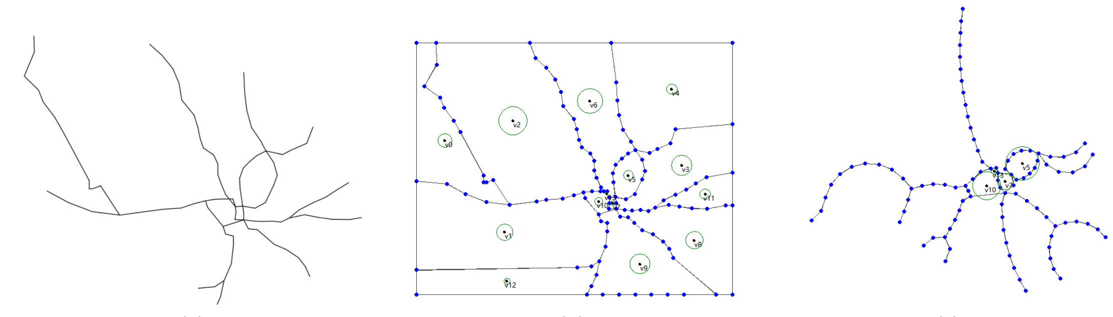

(a) (b) (c)

Figure 6.40: The result of Washington metro map (optimal edge length = 5).

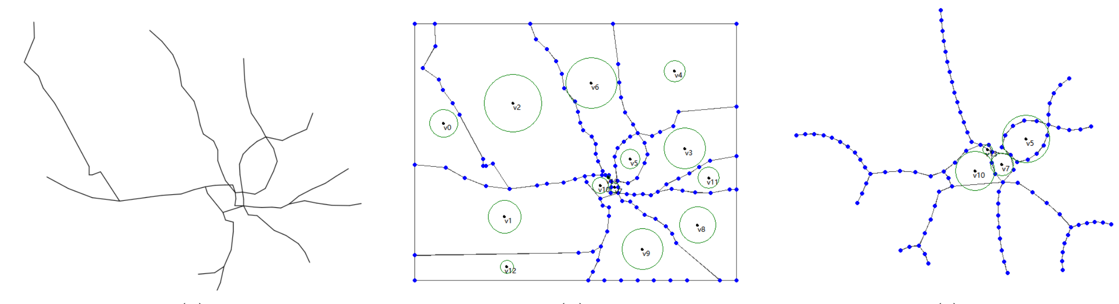

(a) (b) (c)

Figure 6.41: The result of Washington metro map (optimal edge length = 10).

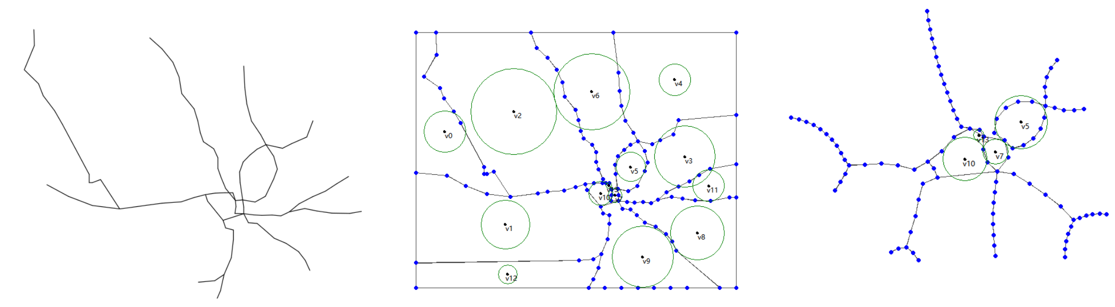
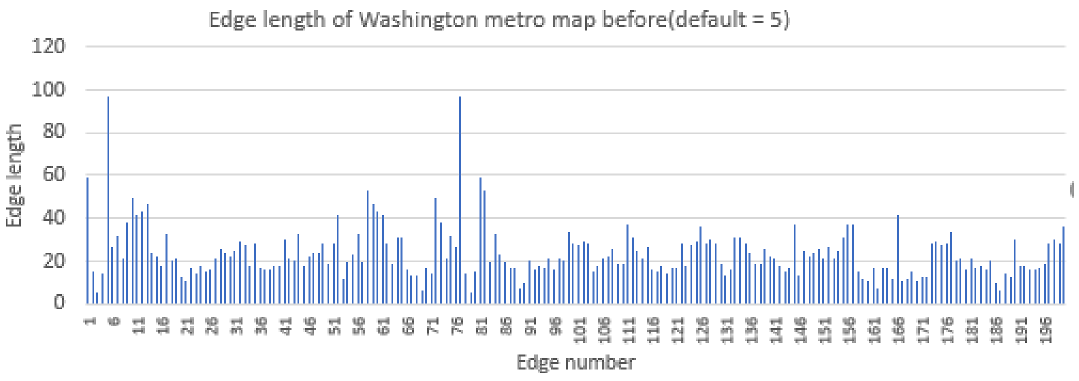
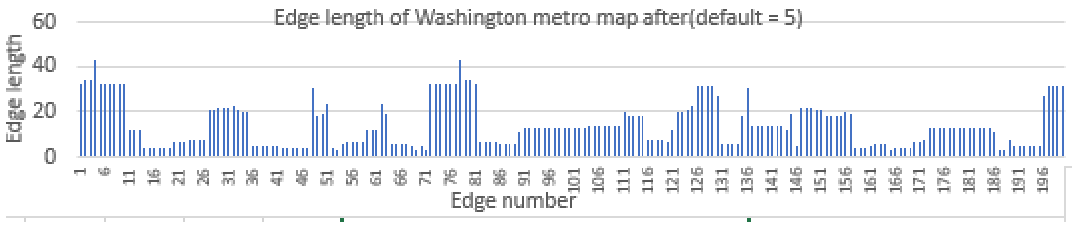

(a) (b) (c)

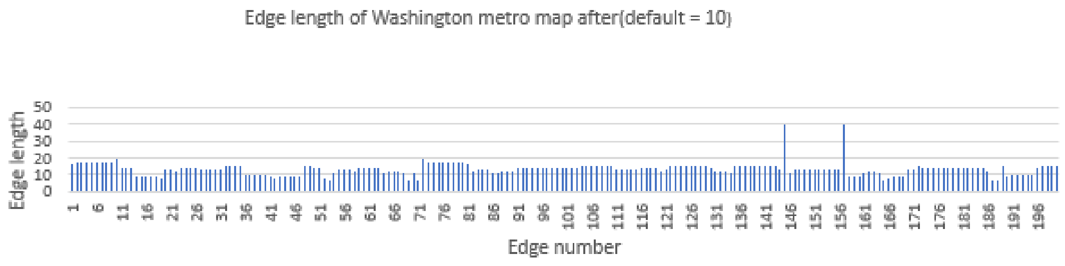
Figure 6.42: The result of Washington metro map (optimal edge length = 15).



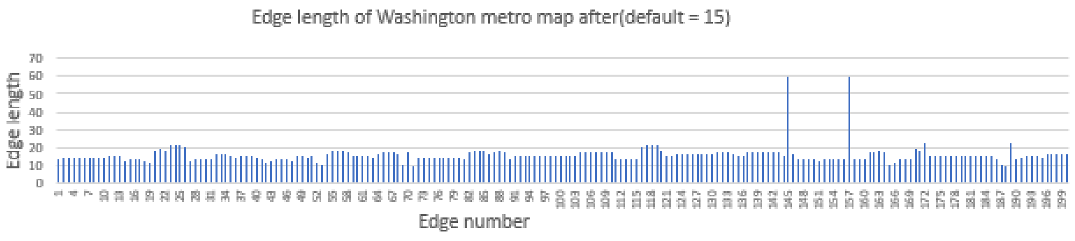
(a) Edge length of Washington metro map before the optimization.



(b) Edge length of Washington metro map after the optimization (optimal edge length = 5).



(c) Edge length of Washington metro map after the optimization (optimal edge length = 10).

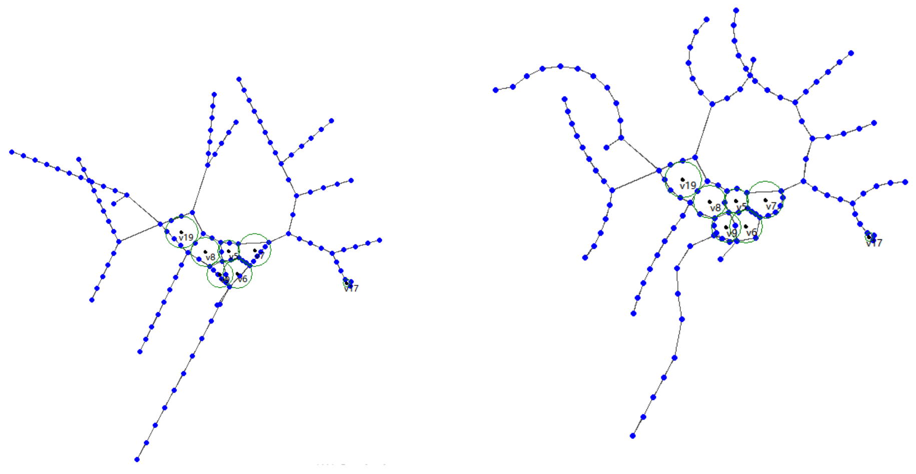


(d) Edge length of Washington metro map after the optimization (optimal edge length = 15).

Figure 6.43: The result of edge balancing of Washington metro map.

6.7.3 Karlsruhe

The three different edge length we choose for Karlsruhe metro map are 5, 10, and 15 respectively. The results are shown in Figure 6.45, Figure 6.46, and Figure 6.47 respectively. All the data is collated into Table 6.5 and Table 6.6. For each edge length, our algorithm can balance the distance between stations and distribute the area of the faces better. The edge length bar chart are shown in Figure 6.48. From the bar chart, the Karlsruhe metro map has been relatively uniform in edge length before optimization, but there are several long edges. After optimization, the long edges are shortened. When the optimal edge length is 5, there exists intersections that will break the topology. This is caused by the three-part-arc (see Figure 6.44). In Figure 6.44a, we reinsert the degree-2 vertices directly with a straight line between two degree-3+ vertices and the topology is not destroyed. However in Figure 6.44b, the curved arcs intersect each other, destroying the topology. But for the left two dangling edges, the three-part-arc can avoid the intersections.



(a) (b)

Figure 6.44: The three-part-arc may destroy the topology and may avoid intersections.

The edge balancing results for Karlsruhe metro map				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	19.02	310.55	185.69	5.82
Edge length = 5	15.33	160.35	65.82	1.99
Edge length = 10	14.24	45.20	46.92	4.24
Edge length = 15	16.58	83.51	69.14	6.71

Table 6.5: The edge balancing results before and after the optimization for Karlsruhe metro map.

The area distribution results for Karlsruhe metro map					
	Average difference	Variance	Maximum	Minimum	Average roundness
Before(edge=5)	569	64246	853	11	0.64
After(edge=5)	28	313	62	7	0.79
Before(edge=10)	182	10942	389	20	0.64
After(edge=10)	118	5047	236	25	0.79
Before(edge=15)	479	69524	855	75	0.65
After(edge=15)	278	25303	488	57	0.79

Table 6.6: The area distribution results before and after the optimization for Karlsruhe metro map.

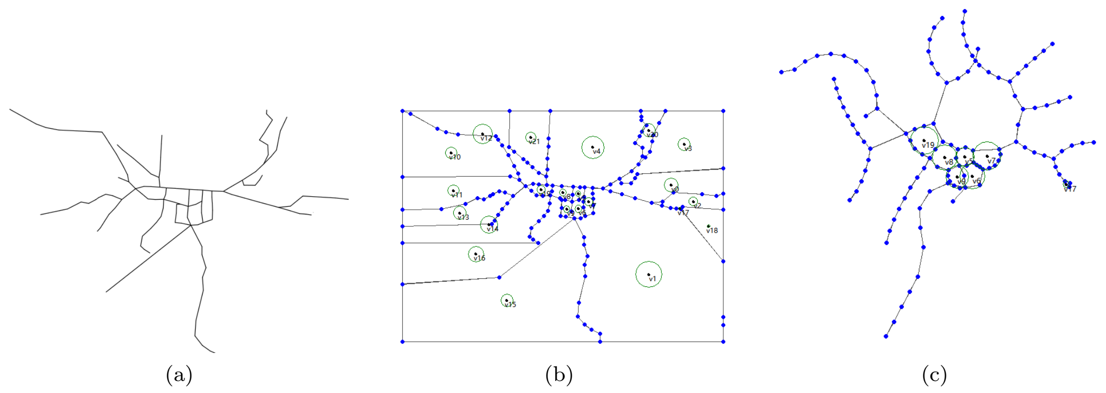

(a) (b) (c)

Figure 6.45: The result of Karlsruhe metro map (optimal edge length = 5).

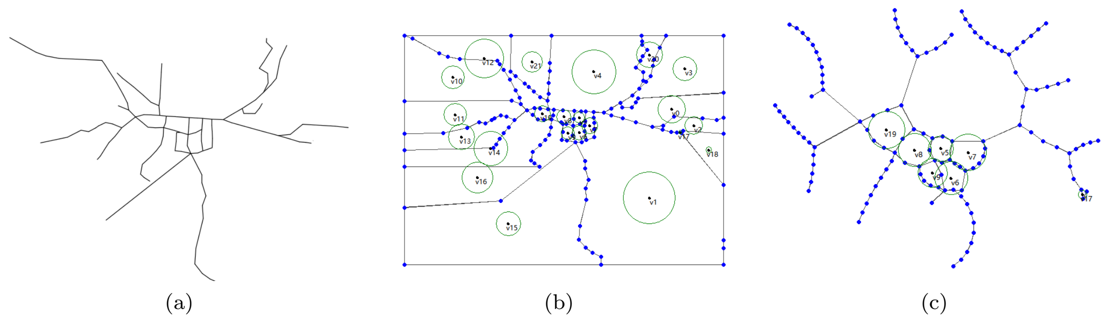

(a) (b) (c)

Figure 6.46: The result of Karlsruhe metro map (optimal edge length = 10).

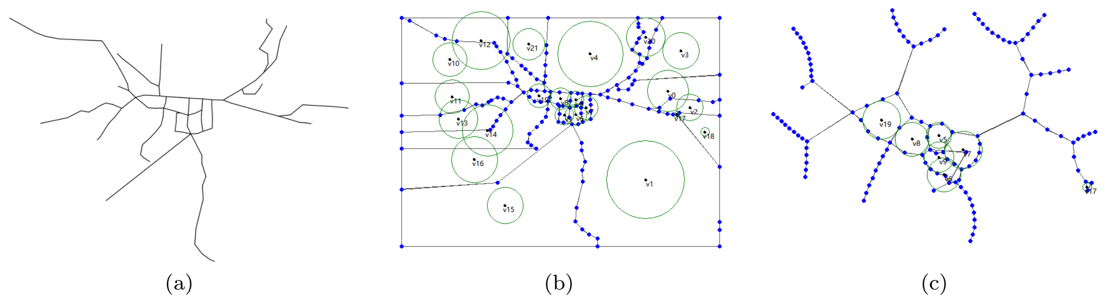
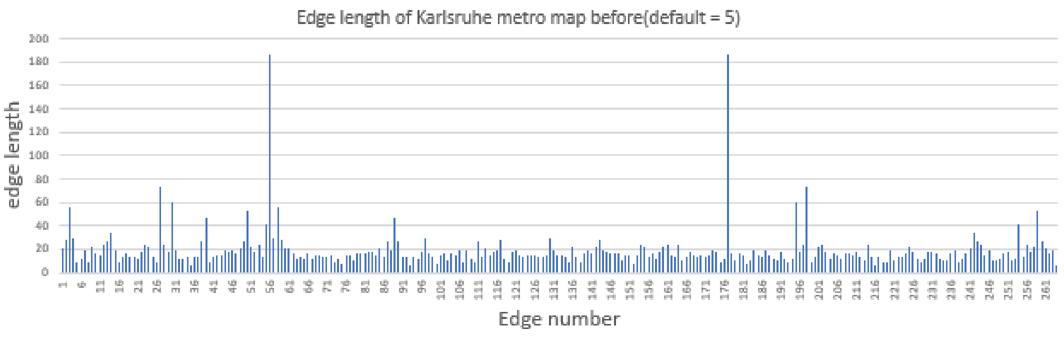
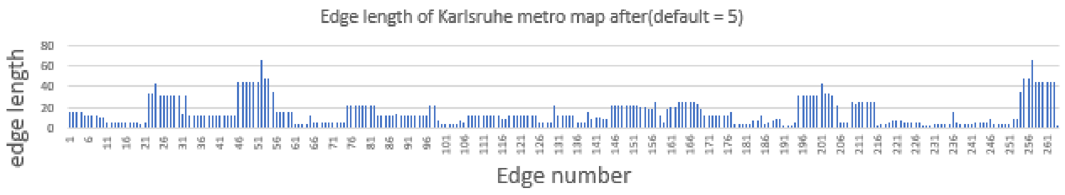

(a) (b) (c)

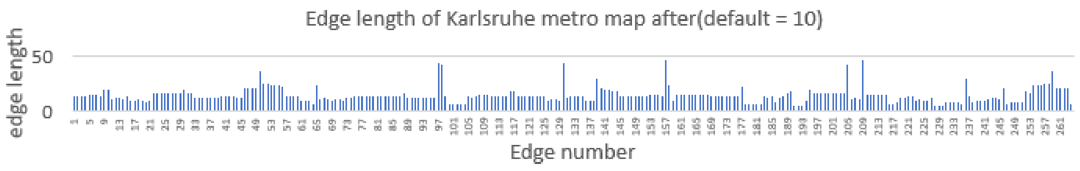
Figure 6.47: The result of Karlsruhe metro map (optimal edge length = 15).



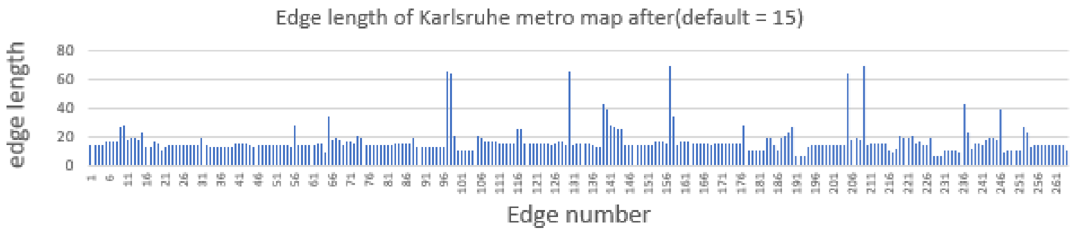
(a) Edge length of Karlsruhe metro map before the optimization.



(b) Edge length of Karlsruhe metro map after the optimization (optimal edge length = 5).



(c) Edge length of Karlsruhe metro map after the optimization (optimal edge length = 10).

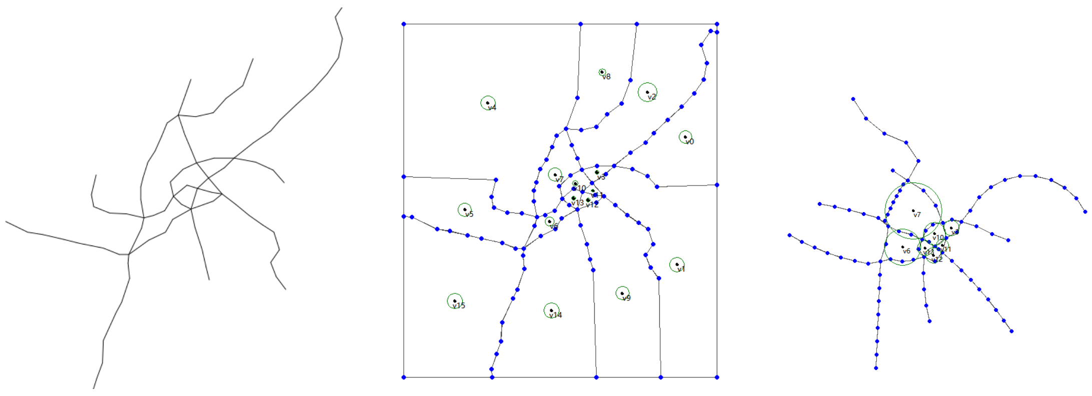


(d) Edge length of Karlsruhe metro map after the optimization (optimal edge length = 15).

Figure 6.48: The result of edge balancing of Karlsruhe metro map.

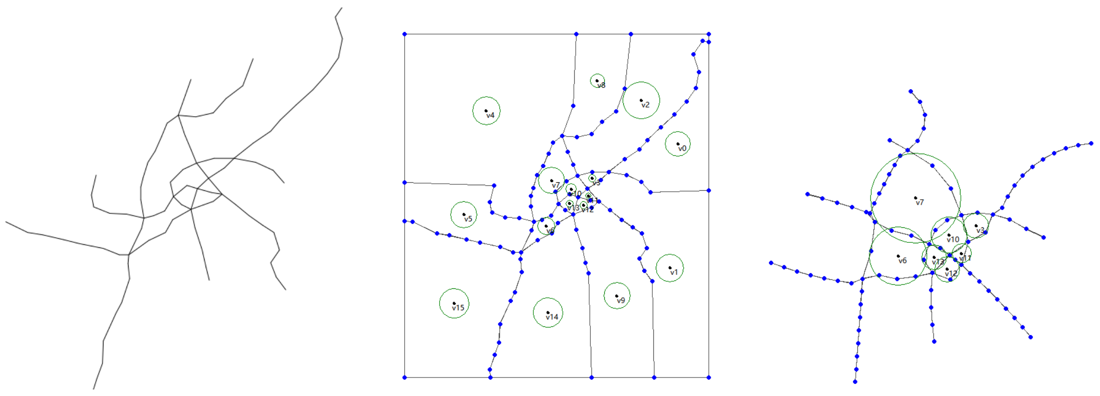
6.7.4 Wien

The three different edge length we choose for Wien metro map are 5, 10, and 15 respectively. The results are shown in Figure 6.49, Figure 6.50, and Figure 6.51 respectively. All the data is collated into Table 6.7 and Table 6.8. The edge length bar chart are shown in Figure 6.52. Our algorithm also performs good in balancing edge length and distributing area in Wien metro map.



(a) (b) (c)

Figure 6.49: The result of Wien metro map (optimal edge length = 5).



(a) (b) (c)

Figure 6.50: The result of Wien metro map (optimal edge length = 10).

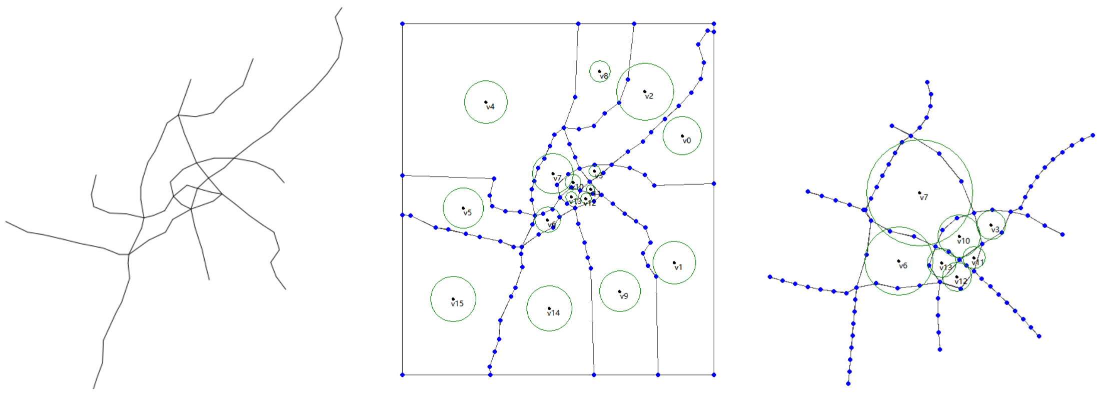

(a) (b) (c)

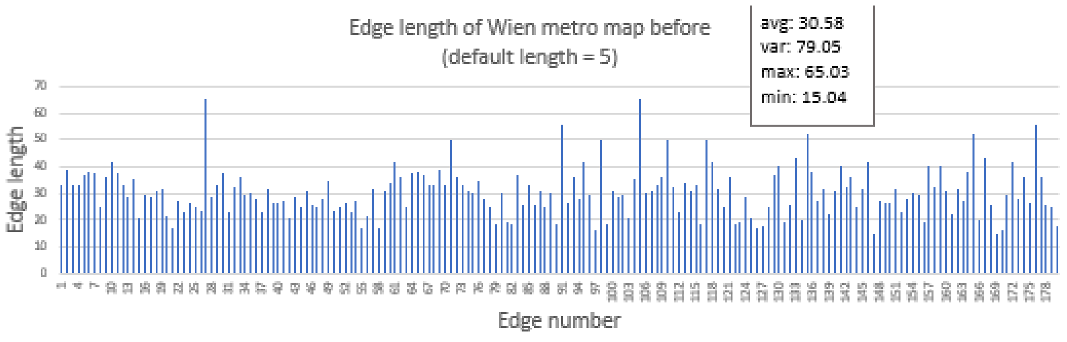
Figure 6.51: The result of Wien metro map (optimal edge length = 15).

The edge balancing results for Wien metro map				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	30.58	79.05	65.03	15.04
Edge length = 5	6.41	3.76	11.18	2.16
Edge length = 10	9.29	8.24	17.9	4.46
Edge length = 15	12.52	24.51	26.76	6.73

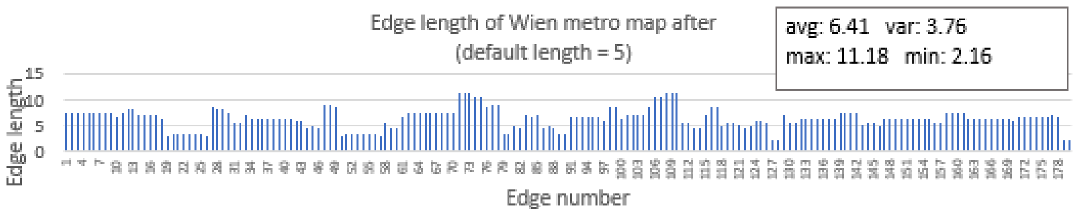
Table 6.7: The edge balancing results before and after the optimization for Wien metro map.

The area distribution results for Wien metro map					
	Average difference	Variance	Maximum	Minimum	Average roundness
Before(edge=5)	2233	5702790	7833	466	0.59
After(edge=5)	25	1640	124	4	0.74
Before(edge=10)	1784	3396198	6128	391	0.59
After(edge=10)	98	25953	492	24	0.73
Before(edge=15)	1036	898108	3286	266	0.59
After(edge=15)	211	127222	1083	30	0.73

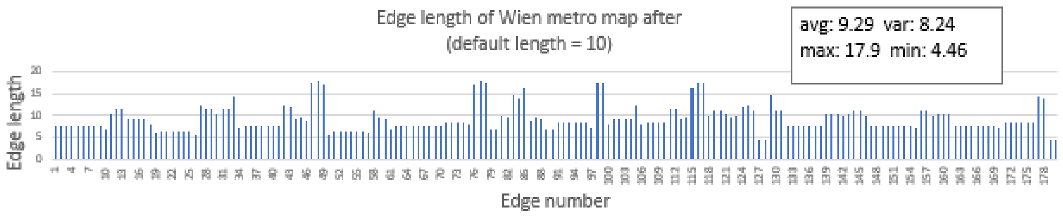
Table 6.8: The area distribution results before and after the optimization for Wien metro map.



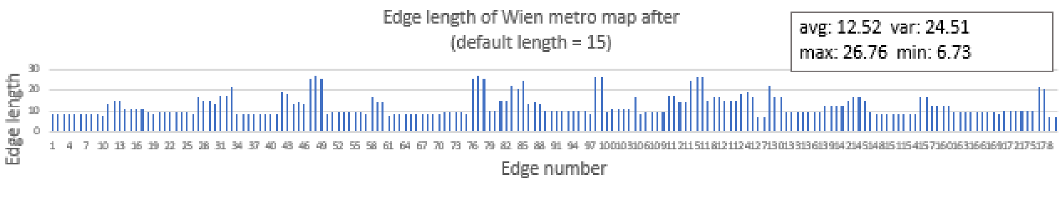
(a) Edge length of Wien metro map before the optimization.



(b) Edge length of Wien metro map after the optimization (optimal edge length = 5).



(c) Edge length of Wien metro map after the optimization (optimal edge length = 10).

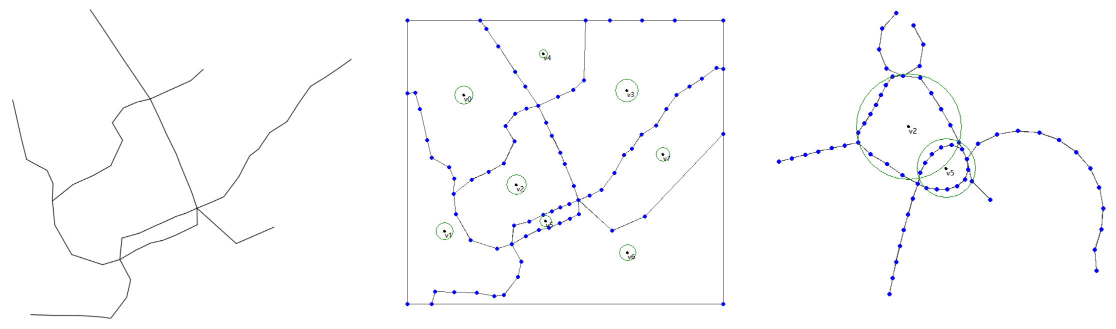


(d) Edge length of Wien metro map after the optimization (optimal edge length = 15).

Figure 6.52: The result of edge balancing of Wien metro map.

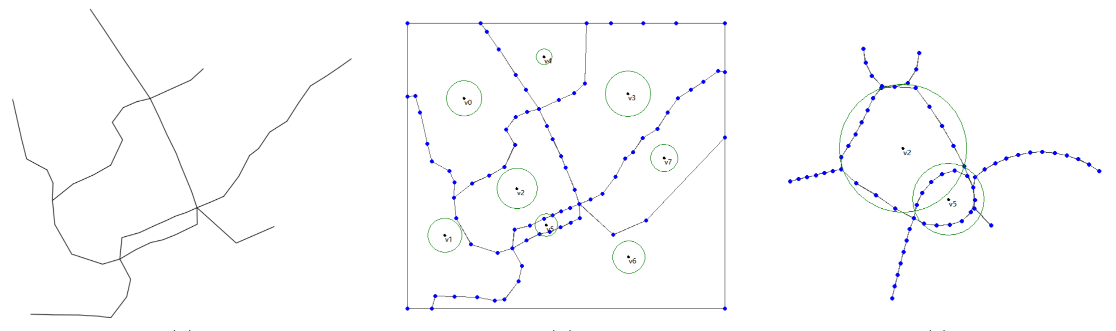
6.7.5 Montreal

The three different edge length we choose for Montreal metro map are 5, 10, and 15 respectively. The results are shown in Figure 6.53, Figure 6.54, and Figure 6.55 respectively. All the data is collated into Table 6.9 and Table 6.10. The edge length bar chart are shown in Figure 6.56.



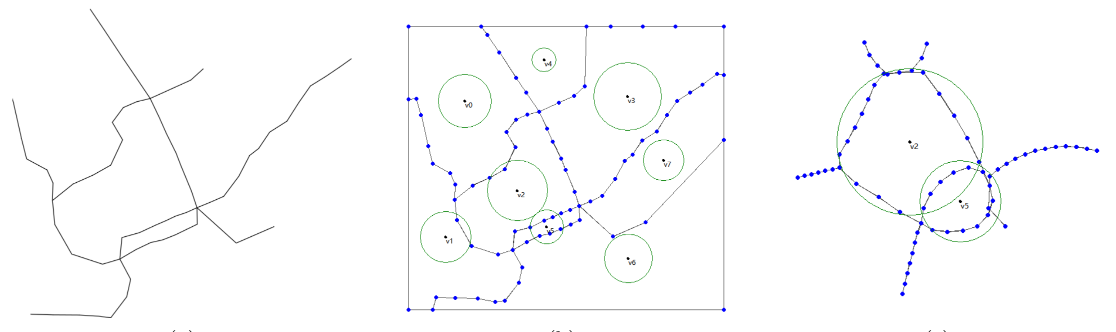
(a) (b) (c)

Figure 6.53: The result of Montreal metro map (optimal edge length = 5).



(a) (b) (c)

Figure 6.54: The result of Montreal metro map (optimal edge length = 10).



(a) (b) (c)

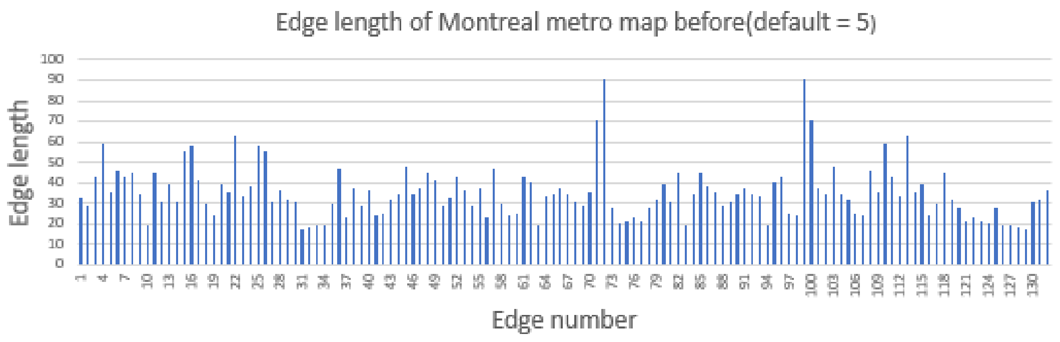
Figure 6.55: The result of Montreal metro map (optimal edge length = 15).

The edge balancing results for Montreal metro map				
	Average edge length	Edge length variance	Longest edge	Shortest edge
Before	35.29	165.25	90.02	17.81
Edge length = 5	12.54	86.55	34.86	3.70
Edge length = 10	12.05	10.84	21.18	7.37
Edge length = 15	14.63	11.27	26.55	10.47

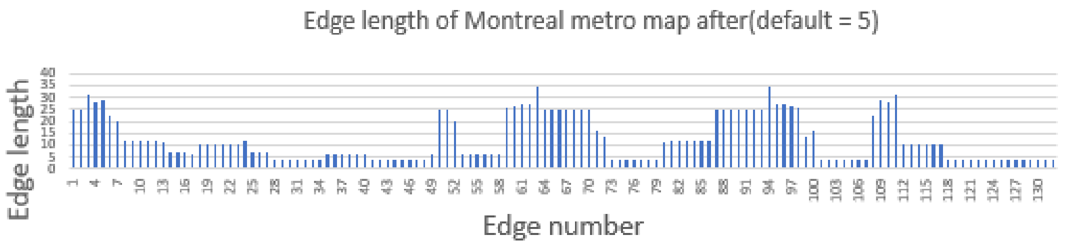
Table 6.9: The edge balancing results before and after the optimization for Montreal metro map.

The area distribution results for Montreal metro map					
	Average difference	Variance	Maximum	Minimum	Average roundness
Before(edge=5)	18924	240713724	34438	3409	0.51
After(edge=5)	291	18576	427	154	0.76
Before(edge=10)	16494	202627433	30728	2259	0.51
After(edge=10)	1153	300739	1702	605	0.77
Before(edge=15)	12443	146434622	24544	342	0.51
After(edge=15)	2566	1545666	3809	1323	0.77

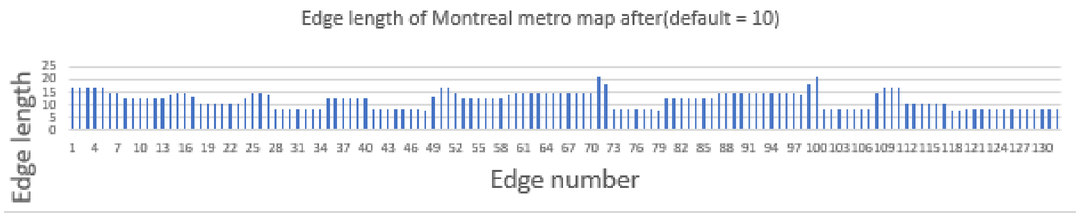
Table 6.10: The area distribution results before and after the optimization for Montreal metro map.



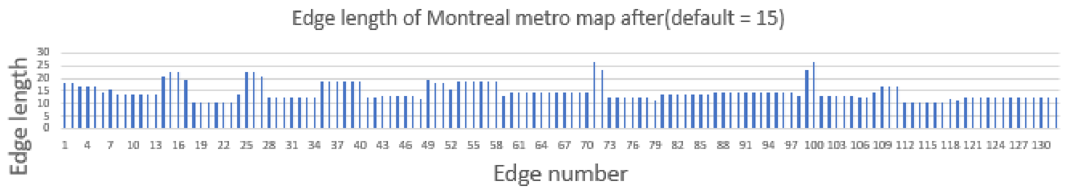
(a) Edge length of Montreal metro map before the optimization.



(b) Edge length of Montreal metro map after the optimization (optimal edge length = 5).



(c) Edge length of Montreal metro map after the optimization (optimal edge length = 10).



(d) Edge length of Montreal metro map after the optimization (optimal edge length = 15).

Figure 6.56: The result of edge balancing of Montreal metro map.

Chapter 7

Conclusions and future work

In this final chapter we make a conclusion of our research and point out directions for the further work of 'face-based' approach for metro map optimization.

7.1 Conclusion

We propose three different 'face-based' approaches for automatic metro-map generation. We describe the algorithm of each approach and do the implementation. The first two approaches do not work well, but they help us find the final direction of research and the final approach can generate readable metro maps.

The first approach is Grid Map approach. We use the grid map and the octilinear layout design criteria. We want to leave more space for labels and avoid clusters of stations, so we decide to increase the roundness of the faces. We draw the $FaceCircle$ with the centroid as the center and aim to move the vertices onto the circle. The vertices inside the circle are moved away from the centroid and vertices outside the circle are moved toward the centroid. We use a shortest-path-graph for a single face but it can not work for multi faces. So we have to move the vertices one by one. Every vertex can be moved to other five candidate positions and we only choose the positions that can enlarge the roundness the face. We have to preserving the topology while moving a vertex and have to keep the octilinear layout. However, for some vertices, none of the five candidate positions can be used so these vertices can not be optimized at all. The shape of the face will still be irregular. Thus, we change our explore direction.

The second approach is Attach Circle approach. We use schematised curvilinear layout in this approach. We attach the vertices on a face evenly onto the circle. Every vertex has a target point on the circle and the vertex is moved by a small step to close the target point. When we move the vertices, we have to keep the topology. This approach can increase the roundness of the faces, but there are long edges in this approach. When two adjacent faces are too far away, their common vertices will fall near the middle of the two faces. These vertices are far from their target points so the connection edges are very long. This makes the metro map more unbalanced. Therefore, we propose the third method to deal with the long edges.

The final approach is a face-based Force-Directed approach. We first rearrange the centroids to get a better centroid layout. We use the force-directed method to achieve this. Then we first remove the degree-2 vertices and just place the degree-3+ vertices. The distance from the adjacent centroids to the degree-3+ vertices are proportional to the number of vertices on the faces. Then we reinsert the degree-2 vertices on the chains between the degree-3+ vertices. For inside degree-2 vertices, we use the apollonian circle arc with two smooth connection arcs. For outside degree-2 vertices, we use the $FaceCircles$ arc and also with two smooth connection arcs. To obtain a real metro map, we add a box around the metro map. The dangling edges are then included in a face and can be processed with our algorithm. The results of this approach shows that our algorithm can improve the readability and usability of the metro maps. The variance of the edge length is reduced and the average edge length is closer to the optimal edge length, meaning that the the distances between the stations are more even. The differences between the faces area and their optimal area decrease, which is also in line with our expectation: the area is proportional to the number of vertices on the face. Finally, we get a readable metro map for each city.

7.2 Future Work

We are the first explore the face-based metro map generation algorithm and we think there are still some shortcomings in our work. Here are a few possible research directions in the future work.

First, for a metro map, preserving the topology is a criteria that must be meet. However, our final approach does not preserve the topology. In the future work, we consider adding another force on each station to preserve the topology. When a station is detected to break the topology, the added force will pull the station in a direction that does not break the topology.

Second, we reinsert the degree-2 vertices onto a three-part-arc. But sometimes these three arcs are too curved to make the route irregular. We want to find other methods that can reinsert the degree-2 vertices in future work.

Third, when we rearrange the centroids, we may change the original spatial orientation of the metro maps. For example, a face that was at the bottom left of another face may be adjusted to the top right. This is the opposite of the geographical impression in people's minds. Therefore, in our future work, we should try to keep the adjusted map in line with people's impression of geography.

Forth, different edge length of the regular polygon will influence the final results. Some edge lengths cause the intersections in the metro map and break the topology, while others can give a most balanced metro map. In the future work, we want to explore a method to find the optimal edge length for each metro map.

Finally, in order to handle the dangling edges, we add a box outside the entire metro map and manually connect the end of the dangling edges to the border of the box. We want to find a better method to deal with edges instead of drawing by hand. Besides, the faces that are adjacent to the outside face will not squeezed to achieve maximum roundness.

Bibliography

[1] Design is history. http://www.designishistory.com/1960/massimo-vignelli/. 2

[2] A fun london metro map from flickr. https://www.flickr.com/photos/fdansv/6099930985/sizes/l/in/photostream/. 6

[3] Graph visualisation powerpoint presentation. https://www.slideserve.com/velika/graph-visualisation-powerpoint-ppt-presentation. 8

[4] Harry beck's original london underground map... but with 2020's tube network. https://londonist.com/london/transport/modern-tube-map-harry-beck-1931-1933. 4

[5] The history of the tube map. https://londonist.com/2016/05/the-history-of-the-tube-map. 4

[6] The history of the tube map. https://londonist.com/2016/05/the-history-of-the-tube-map. 4

[7] The life of the london underground. https://www.xinghuozhiku.com/35030.html. 4

[8] A lost design for a nyc subway map, rediscovered. https://www.bloomberg.com/news/articles/2015-03-11/a-lost-design-for-a-nyc-subway-map-rediscovered. 2

[9] Mini metros – 220 shrunken simplified transit system maps. https://brillianttrains.com/mini-metros/. 1

[10] Unconventional lines diagram. https://ilyabirman.ru/projects/moscow-metro/rectilinear/. 6

[11] Silvania Avelar and Matthias Müller. Generating topologically correct schematic maps. *Technical report*, 336, 2000. 15

[12] Thomas Barkowsky, Longin Jan Latecki, and Kai-Florian Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, page 41–53, Berlin, Heidelberg, 2000. Springer-Verlag. 7, 15

[13] Hannah Bast, Patrick Brosi, and Sabine Storandt. Metro maps on octilinear grid graphs. In *Computer Graphics Forum*, volume 39, pages 357–367. Wiley Online Library, 2020. 16

[14] Hannah Bast, Patrick Brosi, and Sabine Storandt. Metro maps on flexible base grids. In *17th International Symposium on Spatial and Temporal Databases*, pages 12–22, 2021. 16

[15] William Cartwright and Kenneth Field. Beck to the future: time to leave it alone. In *Proceedings of the 26th International Cartographic Conference*, 2013. 2, 5

[16] Daniel Chivers and Peter Rodgers. Octilinear force-directed layout with mental map preservation for schematic diagrams. In *International Conference on Theory and Application of Diagrams*, pages 1–8. Springer, 2014. 7

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

[18] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991. 14, 41, 42

[19] Seok-Hee Hong, Damian Merrick, and Hugo AD Do Nascimento. The metro map layout problem. In *International Symposium on Graph Drawing*, pages 482–491. Springer, 2004. 7, 15, 45

[20] Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *ArXiv*, abs/1201.3011, 2012. 14

[21] Dieter Lutz, Thomas van Djik, and Alexander Wolff. Realtime linear cartograms using least-squares optimisation. 2014. 16

[22] Martin Nöllenburg. *Automated drawing of metro maps*. Universität Karlsruhe, Fakultät für Informatik, 2005. 7, 12, 15

[23] Martin Nöllenburg and Alexander Wolff. A mixed-integer program for drawing high-quality metro maps. In *International Symposium on Graph Drawing*, pages 321–333. Springer, 2005. 15

[24] Mark Ovenden. *Metro maps of the world*. Capital Transport, 2005. 12

[25] Jonathan Stott. *Automatic layout of metro maps using multicriteria optimisation*. PhD thesis, Kent University, 2008. 13

[26] Jonathan Stott, Peter Rodgers, Juan Carlos Martinez-Ovando, and Stephen G Walker. Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):101–114, 2010. 15

[27] Jonathan M Stott and Peter Rodgers. Automatic metro map design techniques. In *Proceedings of the 22nd International Cartographic Conference*, 2005. 15, 18

[28] Hsiang-Yun Wu, Benjamin Niedermann, Shigeo Takahashi, Maxwell J Roberts, and Martin Nöllenburg. A survey on transit map layout–from design, machine, and human perspectives. In *Computer Graphics Forum*, volume 39, pages 619–646. Wiley Online Library, 2020. 3, 5, 6, 16