Automated Drawings of Metro Maps[1]

Martin Nöllenburg[2]

Technical Report 2005-25

September 2005

Fakultät für Informatik, Universität Karlsruhe

[1]This technical report is nearly identical to my Master's thesis [Nöl05].

[2]Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. Supported by grant WO 758/4-2 of the German Science Foundation (DFG).
WWW: i11www.ira.uka.de/algo/group

Abstract

This work investigates the problem of drawing metro maps which is defined as follows. Given a planar graph G of maximum degree 8 with its embedding and vertex locations (e.g. the physical location of the tracks and stations of a metro system) and a set $\mathcal{L}$ of paths or cycles in G (e.g. metro lines) such that each edge of G belongs to at least one element of $\mathcal{L}$, draw G and $\mathcal{L}$ *nicely*. We first specify the niceness of a drawing by listing a number of *hard* and *soft* constraints. Then we show that it is NP-complete to decide whether a drawing of G satisfying all hard constraints exists. In spite of the hardness of the problem we present a mixed-integer linear program (MIP) which always finds a drawing that fulfills all hard constraints (if such a drawing exists) and optimizes a weighted sum of costs corresponding to the soft constraints. We also describe some heuristics that speed up the MIP and we show how to include vertex labels in the drawing. We have implemented the MIP, the heuristics and the vertex labeling. For six real-world examples we compare our results to official metro maps drawn by graphic designers and to the results of previous algorithms for drawing metro maps.

Contents

1 Introduction **1**

2 Preliminaries **7**

2.1 Graphs and Their Drawings 7

2.1.1 Graphs 8

2.1.2 Graph Drawing 8

2.2 Combinatorial Optimization 11

2.2.1 Linear Programming 11

2.2.2 Mixed-Integer Programming 12

3 Related work **13**

3.1 Schematic Maps 13

3.2 Metro Maps 15

3.3 Graph Labeling 16

4 Drawing Metro Maps **19**

4.1 What is a Metro Map? 19

4.2 Metro-Map Esthetics 20

4.2.1 Graphical Criteria 20

4.2.2 Layout Criteria 21

4.3 Modeling Metro Maps 22

4.3.1 Hard Constraints 22

4.3.2 Soft Constraints 23

4.4 The Metro-Map Layout Problem 23

5 Complexity of Drawing Metro Maps **29**

5.1 Relationship to Orthogonal Layouts 29

5.1.1 Rectilinear Graph Drawing 29

5.1.2 Extension to Octilinear Layouts 30

5.2 NP-Completeness of MetroMap 31

5.2.1 Planar 3-Sat 31

5.2.2 MetroMap is NP-Complete 32

6 A Mixed-Integer Program for Metro-Map Layouts **39**
6.1 Linear Constraints . . . 39
6.1.1 Coordinate system . . . 40
6.1.2 Octilinear Edges . . . 40
6.1.3 Preservation of the Embedding . . . 43
6.1.4 Planarity . . . 44
6.2 Objective Function . . . 44
6.2.1 Minimizing Edge Lengths . . . 45
6.2.2 Avoiding Line Bends . . . 45
6.2.3 Preserving Relative Positions . . . 47
6.2.4 Summary of the Model . . . 47
6.3 Speed-Up Heuristics . . . 48
6.3.1 Reducing the Graph Size . . . 48
6.3.2 Planarity Heuristics . . . 49
6.4 Label Placement . . . 50
6.5 Implementation . . . 52
6.5.1 Generating the MIP . . . 52
6.5.2 Optimizing the MIP . . . 53
6.5.3 Graphical Output . . . 54

7 Experimental Results **55**
7.1 Montreal . . . 55
7.2 Vienna . . . 59
7.3 Karlsruhe . . . 61
7.4 Sydney . . . 64
7.5 London . . . 69
7.6 S-Bahn RheinNeckar . . . 72

8 Final Remarks **77**
8.1 Conclusion . . . 77
8.2 Outlook . . . 78

References **79**

Chapter 1

Introduction

Metro maps are ubiquitous in almost all developed urban areas with a (rail-based) public transport system and they "have become the most memorized cartographic items in the world" according to Ovenden [Ove03]. Ever since Harry Beck devised the first and overwhelmingly embraced[1] schematic diagram[2] for the London Underground in 1933 [Gar94] designers and cartographers all over the world adapted his ingenious idea, more or less successfully, to their needs. The longevity of Beck's diagram (it has only changed marginally over the last 70 years) proves its success. Even artists were inspired by the tube map, see for example the poster for an art exhibition in Figure 1.1. However, designing a schematic map today is still a very tedious task. Although schematic maps are usually no longer drawn by hand as Beck did, computers only assist designers by providing draughting software. The decisions where to put stations and lines in the diagram is entirely left to the experience of the cartographer or graphic designer. And this is not at all a simple task, especially in larger and densely connected networks. In this work we investigate an algorithmic method for automatically drawing metro maps. Certainly, the goal is not to generate print-ready diagrams in practice but to automate one step of map production and leave the artistic finishing to professional designers.

Before we can concentrate on actually drawing metro maps we have to understand how a good metro map in the Beck style works and what its main features are. In an interview with Ken Garland, director of the London Transport Museum, Beck stated that "if you're going underground, why do you need to bother about geography? It's not so important. Connections are the thing." [Had03] This keynote is still the guiding principle in drawing metro maps. The purpose of a metro map is to ease navigation on the network for passengers. And passengers want to quickly answer questions like: How do I get from A to B? Where do I have to change trains? How many stops are left to my destination? Exact geography is not only unnecessary for answering these kinds of questions, it is even hindering. Hence, the form of a metro map has to follow its function. The map should be as readable and clear as possible concentrating on the network topology and without displaying unnecessary details. That's why Beck, an engineering draughtsman and Underground commuter himself, designed his diagram conforming to a set of rules: Transport lines are straightened and restricted to horizontals, verticals and diagonals at 45° (we will call such a layout *octilinear*). The scale in crowded downtown areas is larger than in the less dense suburbs in order to create more uniform distances between

[1] 850.000 pocket folders were spread in only two months.

[2] Beck denoted his work as a diagram to distinguish it from previous railway maps. However, we will use these terms synonymously.

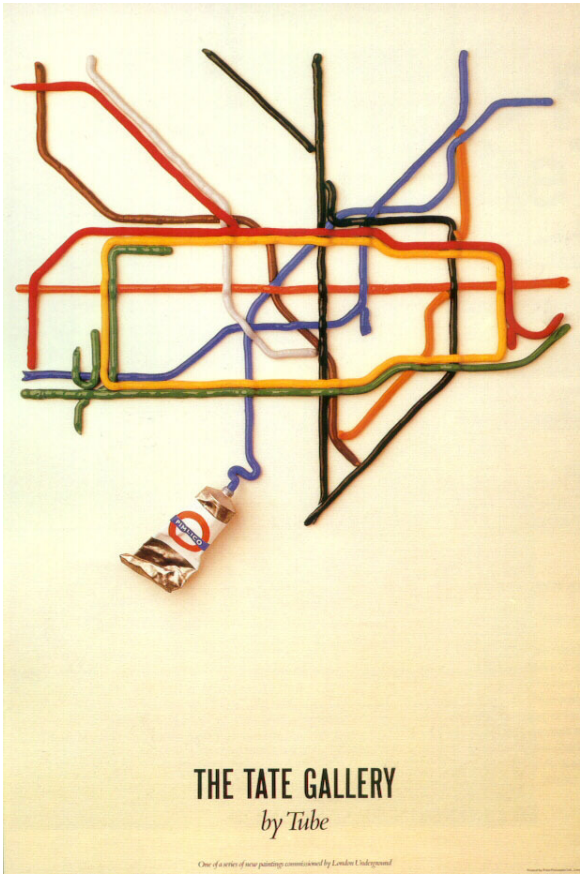


Figure 1.1: The Tube Map as an artwork by David Booth.

adjacent stations. In spite of all distortion, the network topology and a general sense of the geometry, e.g. a certain relative position between metro stations, must be retained. These principles apply to the majority of real-world metro maps as can be verified in Ovenden's book [Ove03] or Morrison's article [Mor96].

In graph-drawing terms the algorithmic task in drawing metro maps is as follows. The transport network can naturally be represented as a graph, where vertices correspond to stations and edges correspond to direct connections between the incident stations. The true location of these geographic entities determines the input layout of the network. This layout is usually planar (otherwise we planarize it) and hence defines an input embedding which has to be preserved by the metro-map layout. The challenge for the algorithm is basically to find vertex positions in the plane such that all edges are drawn as straight *octilinear* line segments and the input topology is preserved. Esthetic criteria such as a small number of bends along the individual metro lines, or a rough preservation of the input geometry need to be optimized over the set of all octilinear, topology-preserving layouts. Algorithmically, we do not bother about design aspects that determine the graphical representation of vertices and edges in the drawing. This notion of a metro-map layout is an interesting compromise between drawing schematic road maps [CBD+01] where vertex positions are (mostly) fixed and "conventional" graph drawing where vertices can go anywhere. The first approach maximizes maintenance of the user's mental map, the second approach maximizes esthetics regardless of the input geometry, which is not given in conventional graph drawing.

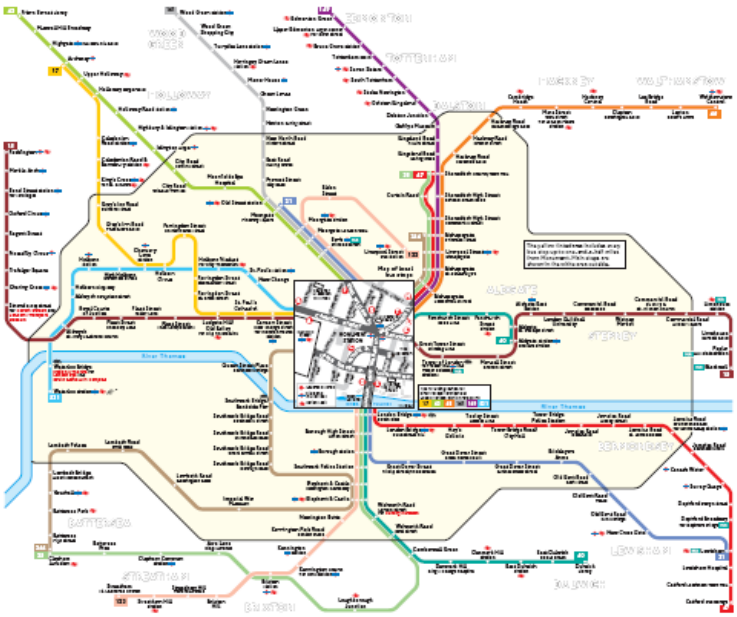


Figure 1.2: A spider map for the metro station *Monument* in London [Trab]. Bus lines are drawn schematically with a geographic inset to display the neigboring bus stops.

Modeling the problem is interesting enough from a theoretic perspective, but one could ask about the practical relevance of automatically drawing metro maps. Do not all metro systems today have already a good map of their network? While this might be the case for some cities, there is an equal number of cities with overly complex schematic or even topographical metro maps. These maps should be improved in the commuters' interest. Current geographic information systems (GIS) might offer drawing geographic metro maps (e.g. the system DIVA[3] can overlay metro lines on the city map) but to the best of our knowledge there is no commercial tool available to automatically draw schematic metro maps. Also, due to the permanent expansion of most metro systems there is a regular need for drawing up-to-date maps. Often, during periods of construction some lines are detoured and temporal maps must be displayed. These temporal maps should of course remain very similar to the regular metro map. In London a relatively new type of maps, so-called *spider maps*, schematically display the bus services of one small district (see Figure 1.2). In the center of the "spider web" the surroundings of one bus stop are shown geographically as an inset to which the schematic lines connect. This inset helps to find the local bus stops. However, once on the bus the geography becomes less important and only topological information is given. Spider maps try to organize the huge bus network of London at least locally. A global bus map, even a schematic one, would be far too complex. These maps must be produced individually for each district, a fact that immediately demands for automatization of the layout process. All these examples show that there is a continuing need for "nice" metro maps.

However the layout principles of metro maps have not only been used in a geographic setting. Sandvad et al. [**?**] and Nesbitt [Nes04] use the *metro-map metaphor* as a way to visualize abstract information related to the Internet and "trains of

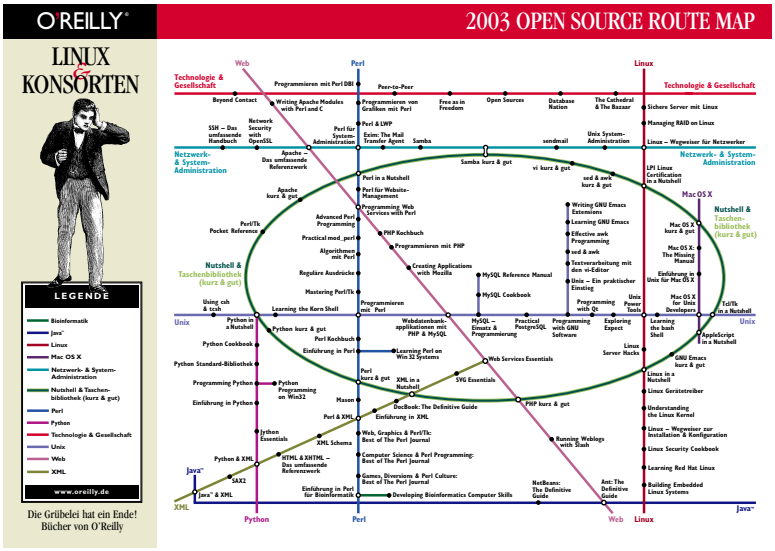
[3]DIVA is a software suite for transport carriers by Mentz Datenverarbeitung GmbH.

thoughts", respectively. Stott et al. [SRB+05] present a prototype tool to draw project plans in a metro-map style. Applications of the metro-map metaphor in practice can be found in Figure 1.3. The upper part of the figure shows the different open source books by the publisher O'Reilly organized as metro lines [O'R]. For example one can interchange between the "Perl" and the "Web" product line at the book "Programming Web Services with Perl". Figure 1.3(b) in the lower part uses the metro-map metaphor to visualize how the proteins of molecular pathways (modeled as metro lines) interact in a cancer cell [HW02].

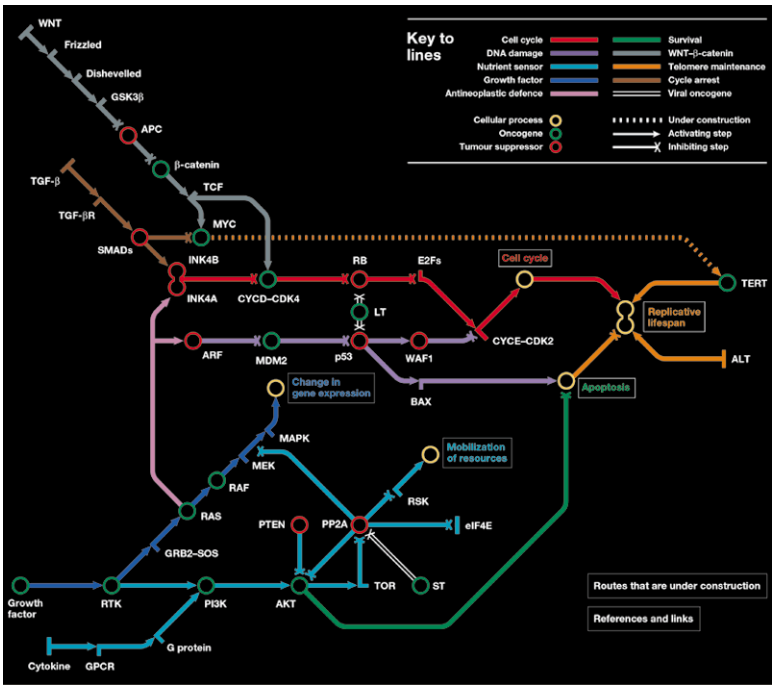
Being a schematic layout style, metro maps or octilinear layouts in general could find more applications in all areas that make use of schematic drawings of graphs. Orthogonal layouts are the predominant layout style for schematic graph drawings in technical and engineering applications today. Examples include circuit diagrams, cable plans, or entity-relationship diagrams. Orthogonal layouts are a very tidy way to visualize these data. Moreover, orthogonal graph drawing has been studied extensively, for an overview see [EFK01] or [dBETT99]. There are several efficient algorithms for orthogonal layouts that optimize various esthetics criteria such as e.g. the number of bends or the area of the drawing. Using octilinear layouts instead could help to avoid bends or save space by making use of the diagonals without losing the tidiness of the diagrams. Depending on the application we might not need to preserve topology, or new esthetics could be formulated. Brandes et al. [BEKW02] introduced the concept of a *sketch* of a graph. A sketch can be handmade or the physical embedding of a geometric network like the real position of telephone cables. Brandes et al. use a path-based min-cost flow formulation to efficiently compute topology-preserving orthogonal drawings of such sketches. Algorithms for octilinear instead of orthogonal layouts in this context would also be of great interest.

There are two previous methods for drawing metro maps by Hong et al. [HMN05] and by Stott and Rodgers [SR04] that are based on similar esthetics. Hong et al. use a modified spring embedder and Stott and Rodgers apply a local optimization technique (see Section 3.2 for more details). However, neither of the methods guarantees an octilinear drawing. This is an important drawback since the tidiness of metro maps depends crucially on the restriction to only four orientations. Our method will formulate the layout problem as a mixed-integer program (MIP) that guarantees a topology-preserving octilinear layout and optimizes a number of esthetics criteria. It is the first method that ensures octilinearity of the drawing. Being a *global* optimization method, the MIP approach does not suffer from the problem of local optima. However, the worst-case time to solve a MIP is not polynomial. So, depending on the problem size we need to apply data-reduction heuristics. We will justify using such an NP-hard optimization technique by proving that the metro-map layout problem itself is NP-hard.

The work is organized as follows. Chapter 2 introduces some basic concepts about graphs and in specific about graph drawing. Also, we present mixed-integer programming from a user's perspective. In Chapter 3 we discuss related works about drawing schematic (metro) maps and graph labeling. Chapter 4 formalizes the intuition of a metro map given in this introduction. We address metro-map esthetics and model hard constraints that need to be fulfilled and soft constraint that should hold as tightly as possible. The chapter closes with a definition of the metro-map layout problem. Subsequently, in Chapter 5, we examine the complexity of drawing metro maps in comparison to drawing orthogonal layouts. We show that it is NP-complete, in contrast to the orthogonal case, to determine whether a metro-map drawing (fulfilling the hard constraints) actually exists for a given input graph. Chapter 6, as the main part of our work, describes how we model the previously introduced problem of drawing metro maps as a mixed-integer program. We



(a) The open source product lines of the publisher O'Reilly.



(b) The molecular circuitry of cancer.

Figure 1.3: Two examples of applying the metro-map metaphor to visualize structured information.

delineate how to enforce octilinearity, preservation of the embedding and planarity with linear constraints. The esthetics criteria are modeled as objective functions to minimize. Also, we suggest several heuristic methods to reduce the problem size and speed up the MIP optimization. Obviously, metro maps should show stations labeled with their names. We illustrate how to modify the input graph such that it comprises vertex labels and how the MIP needs to be changed in order to accommodate these labels. The final section of Chapter 6 briefly describes our prototype implementation of the model. In Chapter 7 we show by means of six different real-world examples how our method successfully draws high-quality metro maps. The examples are of increasing complexity. One of our layouts also includes vertex labels. We conclude in Chapter 8 and point out directions for further work.

Chapter 2

Preliminaries

In this chapter we want to review the main concepts in graph drawing and mixed-integer linear programming that will be used in the remainder of this work. Section 2.1 recalls some common terminology in graph theory and graph drawing. In Section 2.2 we address the basics of combinatorial optimization with mixed-integer linear programs.

2.1 Graphs and Their Drawings

Graphs are one of the most ubiquitous data structures in computer science and are in general used to model relationships between entities. The applications range from UML diagrams in software engineering, which model for example the inheritance relationships between classes in an object-oriented program, over social networks displaying relationships within a group of people to geometric networks such as road graphs or metro graphs modelling connections between certain geographic locations. A common theme among all these fields is that the graph provides some important information about the objects of interest and that it is important to visualize this information for its users. Clearly, representing an abstract graph as an adjacency list or matrix contains exactly the same information as its visualization. However, displaying for example a metro map in the form of an adjacency list in a metro station would be anything but a help for the passengers. Therefore, it is crucial for a human user that a graph is visualized the right way. In a nice and clear drawing the user can immediately find the information he is looking for, but a poor visualization of a graph confuses the user and it will take a long time to extract the relevant data from it. This is illustrated in Figure 2.1, where the same graph is drawn nicely and symmetrically on the one hand and quite confusingly with many edge crossings on the other hand. Yet, there is no unique measure of quality as each application emphasizes different aspects of the graph structure. Consequently the quality of a drawing can be judged quite differently depending on its purpose.

Next, we introduce some basic terminology for graphs and graph drawing in general. For the wide range of algorithms for drawing graphs we refer to the graph drawing literature. The book by di Battista et al. [dBETT99] and the tutorial edited by Kaufmann and Wagner [KW01] give more details. Further, the survey on graph drawing by Eades and Mutzel [EM99] is a good introduction to the topic.

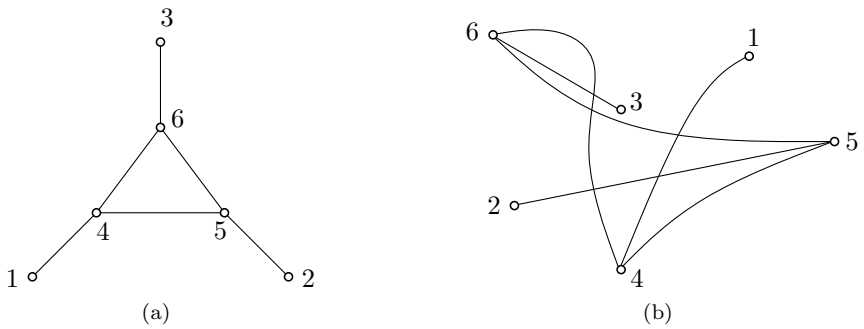


Figure 2.1: Two drawings of the same graph that illustrate how readability can vary between different layouts. In (a) the graph is drawn planarly and symmetrically with straight-line edges whereas in (b) edges are arbitrary curves and there are many edge crossings.

2.1.1 Graphs

An undirected *graph* $G = (V, E)$ consists of a finite set of *vertices* V, $|V| = n$, and a finite set of *edges* E, $|E| = m$, where an edge is an unordered pair $e = \{u, v\}$ of distinct vertices $u, v \in V$. To ease notation we will not use this set notation for edges but simply the short form uv such that $\{u, v\} = uv = vu$ all denote the same undirected edge between vertices u and v. Two vertices that are connected by an edge are called *adjacent*. An edge uv is *incident* to its end vertices u and v as well as to all other edges that are incident to u and v. By $N(v)$ we denote the *neighborhood* of v, i.e. the set of all vertices adjacent to v. Vertices in $N(v)$ are sometimes called *neighbors* of v. The *degree* of a vertex v is the number of edges incident to v and is abbreviated by $\deg(v)$.

A *path* in G is a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ of G such that $v_i v_{i+1} \in E$ for $1 \leq i < k$. The sequence $(v_1, v_2, \ldots, v_k)$ is called a *cycle* if additionally $v_k v_1 \in E$. We call a set $\mathcal{L}$ of paths and cycles a *line cover* of G if each edge in E belongs to at least one element of $\mathcal{L}$. An element $L \in \mathcal{L}$ is called a *line*. Let m' be the total number of edges on all lines in $\mathcal{L}$. Then $m' > m$.

We define *collapsing* a degree-2 vertex as follows. Let $u \in V$ be a vertex with $\deg(u) = 2$ and $N(u) = \{v_1, v_2\}$. Further assume that $v_1 v_2 \notin E$. Then collapsing u results in a graph $G' = (V', E')$, where $V' = V \setminus \{u\}$ and $E' = (E \setminus \{uv_1, uv_2\}) \cup \{v_1 v_2\}$. This operation simply removes u and reconnects its two neighbors with an edge. Since the two neighbors had not been adjacent before, we avoid possible conflicts with multi-edges.

Graphs can convey more information than just the adjacency relation between vertices. For example *weighted graphs* assign a real weight $w(e)$ to each edge e. *Labeled graphs* have vertex and/or edge labels, e.g. a string $\lambda(v)$ with the name of v. More data can be attached to vertices and edges as desired.

2.1.2 Graph Drawing

In general, a (2-dimensional) *drawing* or *layout* Γ of a graph G is a function that maps each vertex $v \in V$ to a distinct point $\Gamma(v) \in \mathbb{R}^2$ and each edge $uv \in E$ to a simple open curve $\Gamma(uv)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. Formally, a graph G and its drawing Γ are different objects. Nevertheless, we often do not distinguish explicitly

between graph and drawing, e.g. we say "e is a straight line segment" instead of "$\Gamma(e)$ is a straight line segment".

A drawing Γ is *planar* if no two edges intersect, except at common end vertices. A graph that admits a planar drawing is called a *planar graph* and a planar graph with maximum vertex degree k is a *k-planar graph*. Planar drawings partition the plane into topologically connected regions, the *faces* of the drawing. There is one unbounded region, which is called the *external* face, all other faces are *internal* faces. Two faces are *adjacent* if their boundaries share an edge and vertices or edges on the boundary of a face are *incident* to this face. The *degree* of a face f, in short $\deg(f)$, denotes the number of edges on its boundary.

For each vertex v in a planar drawing a circular ordering of its incident edges is imposed by their counterclockwise sequence around v. This ordering can also be regarded as an ordering of the set $N(v)$ of neighbors. We say that two planar drawings of the same graph are *equivalent* if they have the same circular orderings of the sets $N(v)$ for all vertices v. An equivalence class for this relation is denoted an *embedding* of G. An alternative but equivalent description of an embedding is the set of all cycles bounding the faces, where one face is specified as the external face. A planar graph together with an embedding is referred to as *embedded graph* or simply *plane graph*.

A general layout of a graph maps edges to simple open curves as stated above. There are classes of drawings where additional conditions must be satisfied by these curves. These classes are often referred to as *drawing conventions*. In a *straight-line drawing* each edge is drawn as the straight line between its end vertices. *Orthogonal* or *rectilinear layouts* force edges to be drawn as polygonal chains of horizontal and vertical line segments. Orthogonal layouts are well studied and are used for example for drawing circuit plans or software engineering diagrams. Their schematic appearance makes rectilinear drawings in a way interesting to apply for drawing schematic maps such as metro maps. Yet, restricting edges to orthogonal polylines is often too limiting for schematic maps. Therefore we introduce the concept of *octilinear* layouts. In an octilinear layout all edges are drawn as chains of horizontal, vertical or 45-degree diagonal line segments. Obviously, octilinear layouts can approximate arbitrary straight-line segments better than rectilinear layouts. Moreover, the maximum possible vertex degree increases from 4 to 8 in octilinear layouts. We discuss the relationship between orthogonal and octilinear layouts in more detail in Section 5.1. All three, straight-line drawings, orthogonal and octilinear layouts, belong to the more general class of *polyline drawings*, which represent edges as arbitrary polygonal chains. See Figure 2.2 for examples of each class of drawings. A drawing convention that can be combined with the above classes is the class of *grid layouts*. In a grid layout all vertices must have integer coordinates.

If an edge is drawn as a polygonal chain with two or more line segments it has *edge bends* where two segments meet. For a polyline drawing of a graph G with associated line cover $\mathcal{L}$ we define *line bends* as follows. A *line bend* on a line $L \in \mathcal{L}$ is either a bend on one of the edges of L or a bend between two consecutive edges of L at their common vertex. Considering the angles in a polyline drawing Γ, the *angular resolution* of Γ is defined as the minimum angle between any two adjacent line segments in Γ. The angular resolution of the graph G is defined as the maximum angular resolution of all its drawings. The angular resolution of orthogonal layouts is 90 degrees while the angular resolution of octilinear layouts is 45 degrees.

Readability of a layout is an application-specific criterion by which humans usually measure the quality of a layout. This is naturally a vague measure and hard to grasp mathematically. Still, various attempts have been made to specify proper-

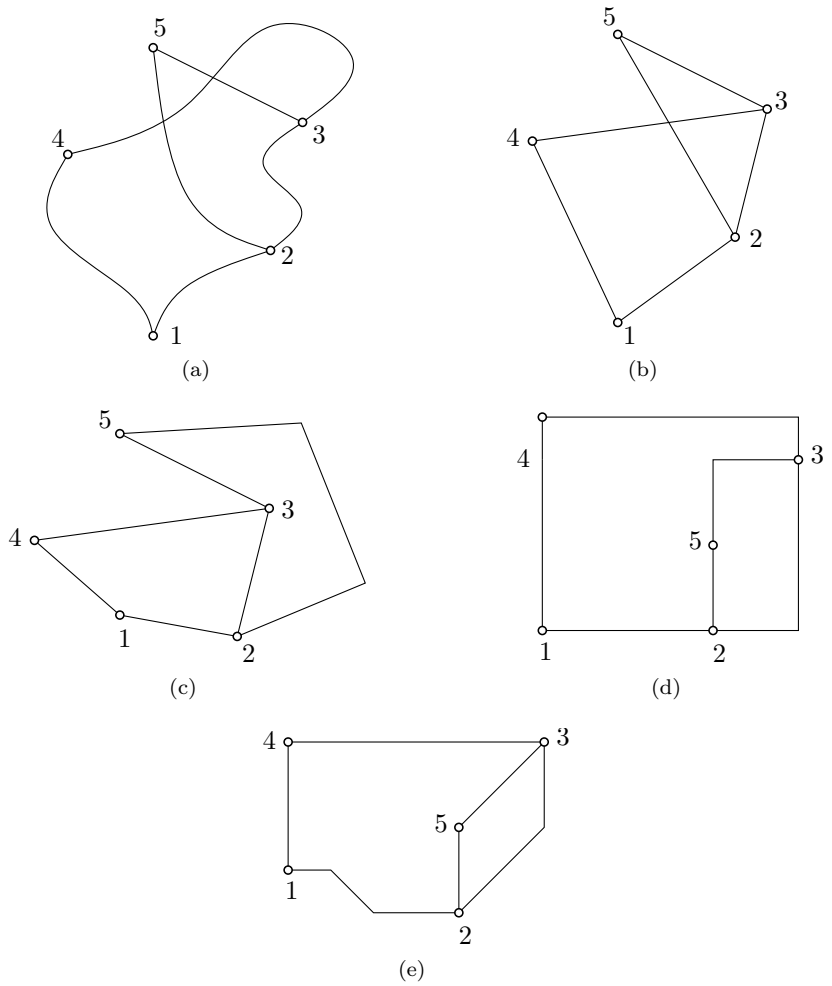


Figure 2.2: Five drawings of the same planar graph G. In (a) the drawing is non-planar and uses simple open curves. Subfigure (b) shows a non-planar layout with straight-line edges. Drawing (c) is a planar polyline layout and (d) is a planar orthogonal drawing. Layout (e) is octilinear. Note that the embeddings in (c) and (d) are different while (d) and (e) have the same embedding.

ties that readable drawings typically show. Commonly accepted *esthetics criteria* (see [dBETT99]) comprise:

- small **number of crossings**, none at all if possible,
- small **drawing area**, which is only meaningful if arbitrary down-scaling is prohibited, e.g. by guaranteeing minimum distances between any two vertices,
- small **total edge length** (under the same conditions as before),
- **uniform edge length**,
- small number of **edge bends**, either for the layout as a whole or per edge,
- large **angular resolution**, and
- as **symmetric** as possible.

These criteria are associated with mostly computationally hard optimization problems and often conflict with each other when applied simultaneously. For example the number of crossings in a drawing could be reduced by allowing edges with more bends. Hence, one must be aware of these trade-offs and carefully select the right esthetics.

Apart from these more structural properties of graph layouts, there are plenty of possibilities how to visualize a single layout. Vertices can be shown in various shapes (squares, circles, ...), sizes and colors. Edges are usually drawn as curves, but they too can have different colors and thicknesses. These graphical properties are undoubtedly important for the readability of a diagram but an algorithmic solution for graph drawing usually just computes vertex positions and the curves representing edges and leaves the decision about graphical features to the user.

For labeled graphs there is the additional requirement to place labels of given size close to their respective vertices or edges. Labels should neither overlap each other nor obscure other edges and vertices. There are two classes of labeling problems. The first problem is to label a given drawing of a graph and the second problem is to draw and label a graph simultaneously so that the required space of the labels is directly considered in the layout. For previous results on graph labeling see Section 3.3 and for an extensive bibliography on map labeling literature see [WS96].

2.2 Combinatorial Optimization

Combinatorial optimization in general means optimization of some target function over a set of discrete choices. In this work we will use an optimization technique called *mixed-integer linear programming* (MIP) to find a metro map layout that optimizes some esthetic criteria over the set of all feasible layouts. To this end, we first introduce *linear programming* (LP) in Section 2.2.1 and then, in Section 2.2.2, the closely related notion of mixed-integer linear programming. This introduction will focus on the user's point of view and not on the extensive theory of linear programming or the algorithms to solve linear programs. For a more general overview of LP and MIP see for example the surveys on linear programming by Chandru and Rao [CR99b, CR99a]. Schrijver's book [Sch86] and Bertsimas and Tsitsiklis [BT97] cover theory and algorithms for (integer) linear programs.

2.2.1 Linear Programming

Linear Programming, abbreviated LP, deals with the minimization or maximization of a linear objective function such that the solution respects a set of linear constraints as follows.

Definition 2.1 *Let $A \in \mathbb{R}^{m \times n}$ be an $m \times n$ matrix, $b \in \mathbb{R}^m$ an m-dimensional vector and $c \in \mathbb{R}^n$ an n-dimensional vector. Then the corresponding* linear program *is given as*

$$\begin{array}{lll} \textit{minimize} & c^T x & \\ \textit{subject to} & Ax & \leq \quad b, \end{array}$$

where $x \in \mathbb{R}^n$ is an n-dimensional vector of real variables.

This is just one form of a linear program. Equivalently, we could maximize the objective function or use $Ax = b$ or $Ax \geq b$ as the linear constraints. These types of linear programs can be transformed into one another. This also means that in

practice we can mix the three different types of constraints in an LP since they are transformable into the desired type.

The linear constraints in an LP define a set of the form $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, called the *feasible region*, which forms a *polyhedron*. Each vector x in the feasible region is denoted as a *feasible solution*. If the feasible region is empty, the LP is called *infeasible*. The polyhedral shape of the feasible region is exploited in algorithms that find an optimal solution but this is beyond the scope of this work.

There is a number of algorithms for the solution of linear programs. The simplex algorithm [Dan51] is often used in practice although it does not have a polynomial worst-case running time. Hačijan [Hač79] developed a polynomial algorithm, the ellipsoid method, which only is of theoretical interest. Some years later Karmarkar [Kar84] published his interior point method, a new polynomial algorithm being competitive in practice as well. Commercial optimization tools such as CPLEX or Xpress-MP implement different LP algorithms and are widely used for industrial optimization tasks.

2.2.2 Mixed-Integer Programming

In many situations an optimization problem involves discrete resources but a linear program is not capable of expressing that some variables can only take discrete values. Hence it seems to be a natural extension of linear programs to restrict the ranges of certain variables. Similar to LP we define *mixed-integer programming* (MIP).

Definition 2.2 *Let $A \in \mathbb{R}^{m \times n}$ be an $m \times n$ matrix, $b \in \mathbb{R}^m$ be an m-dimensional vector and $c \in \mathbb{R}^n$ be an n-dimensional vector. Let $J \subseteq \{1, 2, \ldots, n\}$. Then the corresponding* mixed-integer program *is given as*

$$\begin{array}{lrcll} \textit{minimize} & c^T x & & & \\ \textit{subject to} & Ax & \leq & b & \textit{and} \\ & x_j & \in & \mathbb{Z} & \forall j \in J, \end{array}$$

where $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is an n-dimensional vector of variables.

Again, as in the general LP case, constraints can be stated as either linear equalities or inequalities at the same time. If J is empty, the problem is a regular LP, and if all variables are integers, it is an *integer program*. The *relaxation* of a MIP is the LP that results from dropping the integrality constraints. The matrix and vector notation for an LP or MIP is convenient at this coarse level of detail but when it comes to actually modelling a problem with linear constraints then we will just give a list of the constraints instead of the full rows of the matrix A. Obviously, each constraint can be expanded to a matrix row by using 0 as coefficient for unused variables.

Unlike linear programming, mixed-integer programming is an NP-complete problem and many other classical NP-complete problems can be easily formulated as MIP. Nevertheless, modern MIP solvers such as CPLEX or Xpress-MP often successfully solve hard optimization problems relatively fast in practice. Algorithms for mixed-integer programs usually first solve the LP relaxation of the input problem and then try to reduce the number of remaining fractional variables using a heuristic branch-and-cut search tree algorithm. Problem-specific fine-tuning of the parameters for the MIP solver can largely influence its running time.

Chapter 3

Related work

This chapter is meant to give an overview of the previous work on automated drawing of metro maps. Metro map drawing is a problem that has been studied only recently but is closely related to the general schematization of geographic maps, a subdiscipline of automated cartography. Some additional requirements and rules apply for metro maps. Due to the increased popularity of geographic information systems (GIS), not only in the administration but also for applications on the internet such as route planners, city maps, geographical search etc., there is a high demand for user-adapted map-drawing systems. Currently, certain tasks in map generation can be automated to assist cartographers. When it comes to schematizing maps, research started in the 1970s with works on line simplification, see the survey by Weibel [Wei97]. Current research deals with techniques to schematize maps using only a restricted number of orientations for lines in the drawing. This requirement is similar to what we demand for metro maps.

We stick to the historic development by first characterizing some previous results in map schematization in Section 3.1 and then, in Section 3.2, describing recent approaches for drawing metro maps in particular. Finally, since we combine labeling and graph drawing in this work, Section 3.3 gives related results on drawing labeled graphs.

3.1 Schematic Maps

Neyer [Ney99] studies a line simplification problem where each simplified line segment must be parallel to one of a set of given orientations. More precisely, the input is a polygonal chain P, a set $\mathcal{C}$ of orientations, and a parameter ϵ. The goal is to find another polygonal chain Q which is a $\mathcal{C}$-oriented approximation of P, i.e. the distance between P and Q in the Fréchet metric is at most ϵ, Q uses a minimum number of line segments and each line segment is parallel to an orientation in $\mathcal{C}$. Neyer gives a dynamic programming algorithm to compute $\mathcal{C}$-oriented line simplifications in time $O(kn^2 \log n)$, where n is the number of vertices on P, k is the number of segments of Q and the number of orientations $|\mathcal{C}|$ is constant. The value of ϵ only indirectly influences the time complexity: the smaller ϵ the larger k. She states that, as an example, $\mathcal{C}$-oriented line simplification can be used for schematizing metro lines in a metro map. In that case $\mathcal{C}$ would contain the four orientations *horizontal*, *vertical* and both *diagonals*. Using a reasonable value for ϵ maintains the relative geographical position of stations on the line. However, this

algorithm only considers one path in the metro network at a time. Hence, the interaction between different paths or metro lines in the network is not taken into account. By simplifying one line at a time new intersections might appear or existing intersections might disappear. This not only results in changing the topology of the network, it also can disrupt the graph structure by disconnecting vertices. Decreasing the value of ϵ might anticipate this disadvantage in some cases but it also reduces the ability of the method to simplify paths as they are thus forced to stay closer to their respective input paths.

A further algorithm to schematize maps was presented by Barkowsky et al. [BLR00]. Although designed for drawing schematic maps in general, they also address drawing metro maps with their method. This is to the best of our knowledge the first attempt to automate the problem. They use *discrete curve evolution*, i.e. an algorithm for polygonal line simplification, to treat the lines of the Hamburg subway system. The algorithm preserves the topology of the input map. However, it neither restricts possible edge directions nor does it increase station distances in the crowded downtown area to enhance readability. Hence, the algorithm seems to be more suitable for simplifying road maps than for schematizing public transport maps. Stations are labeled but no effort is made to avoid label overlap.

Avelar and Müller [AM00] implemented an algorithm to modify a given input map by moving the endpoints of line segments such that the resulting map improves in terms of their esthetic criteria. They claim to preserve the map's topology using only simple geometric operations, which is different from previous methods for preserving topology. The goal of their esthetics is to draw edges as straight octilinear line segments. They set a maximum length for each segment and a minimum distance between different segments. Basically, they iteratively calculate for each vertex in the map a new position depending on the positions of its neighbourhood such that edges approach the desired directions. The algorithm was successfully applied to the street network of Zurich. However, not all line segments could be drawn octilinearly because vertex positions are influenced by several potentially conflicting terms. Labels were not included.

An efficient algorithm for schematizing road networks is described by Cabello et al. [CBD+01]. They draw edges of the input network as octilinear paths with two or three links while preserving the input embedding. The user can restrict the links to certain orientations, e.g. horizontal-diagonal-horizontal and vertical-diagonal-vertical paths with angles of 135°. Vertex positions keep their original position. The algorithm runs in $O(n \log n)$ time as long as input edges are drawn monotonously and it detects if no such schematized map exists. In this aspect the algorithm by Cabello et al. is similar to ours because both methods guarantee an octilinear drawing if one exists and they fail otherwise. The basic idea of their algorithm is to compute an order among all input paths and incrementally construct the new drawing according to this order. For drawing metro maps the major disadvantage of this algorithm is that vertices, i.e. metro stations, keep their original positions. In real-world metro maps, however, moving stations is crucial for generating clear and structured layouts. Not doing so leads to many unnecessary bends in the layout and crowded downtown areas remain confusing.

A more recent work by Cabello and van Kreveld [CvK03] covers a combinatorial approach to the problem of aligning points. Given a set of points, the task is to align as many points as possible horizontally, vertically or diagonally, where each point can be placed somewhere in its own, given region. In their paper they consider circles, rectangles or Voronoi cells as the regions around each point. The points represent the vertices of an underlying graph and alignments are only considered for adjacent vertices. Therefore, the alignment problem seems to be an important

task in schematic network design, where one goal is to place vertices such that as many edges as possible are drawn as straight, octilinear line segments while roughly preserving their positions. After modifying the vertex positions, the remaining non-octilinear edges could for example be simplified with the previous method of Cabello et al. [CBD+01]. Cabello and van Kreveld [CvK03] show that maximizing the number of alignments is NP-hard for general planar graphs. They give approximation algorithms for several graph classes, among them a factor $k/(3k+3)$-approximation for planar graphs and convex regions. This approximation requires $n^{O(2^k)}$ time. One inherent disadvantage of their method is that it can move vertices within their regions such that the topology of the input graph changes. It also remains open how well the algorithm works on metro graphs in practice.

The only of the presented methods that guarantees an octilinear and topology-preserving result is the algorithm by Cabello et al. [CBD+01]. However, since it leaves vertex positions untouched, it should not be the method of choice for drawing metro maps. The other methods either preserve the embedding but do not guarantee octilinearity or they generate octilinear drawings but potentially change the topology. Our method overcomes these drawbacks.

3.2 Metro Maps

Two recent publications give algorithms that are explicitly designed to draw metro maps. Hong et al. [HMN05] give five similar methods that are constructed on top of each other. The most refined of these methods modifies a topology-maintaining spring embedder such that edge weights are taken into account and such that additional magnetic forces drag edges towards the closest octilinear direction. Edges are drawn straight-line, without bends. Relative position between vertices is only taken into account implicitly by using the original embedding as initial layout.

In a preprocessing step the metro graph is simplified by removing all degree-2 vertices and reconnecting both their neighbors. Hence, the simplified graph consists only of intersection and degree-1 vertices. Chains of degree-2 vertices are *collapsed* and represented by a single edge. The weight of each remaining edge is set to the number of original edges it represents. After the final layout has been computed, all degree-2 vertices are re-inserted into the corresponding edges in an equidistant manner. This data-reduction step reduces the running time considerably. Station labels are placed in one out of eight directions. While label–label overlaps are avoided, labels sometimes intersect network edges. The results of Hong et al. are clearly superior to those of Barkowsky et al. [BLR00] in the previous section. However, they are still not very similar to commercial maps drawn by graphic designers. The main deficiency is that most edges in the final layouts are close to, but not quite octilinear. This seems to be due to the fact that the magnetic forces that determine the layout are the sum of many conflicting terms. Moreover, edge lengths are far from being uniform. In some of their examples one single edge can be as long as about twenty of the shortest edges. This gives the layout a rather unbalanced look.

Stott and Rodgers [SR04] draw metro maps using multi-criteria optimization based on hill climbing. For a given layout they define metrics for evaluating the number of edge intersections, the octilinearity and the length of edges, the angular resolution at vertices and the straightness of metro lines. The score of a layout is the sum over these five metrics, the lower the score the higher the quality. Their optimization process is iterative. They start with a layout on the integer grid that is obtained from the original embedding. In each iteration they loop through all vertices. For each vertex they consider alternative grid positions within a certain

radius that shrinks with each iteration. For each of these grid positions they compute the quality of the modified layout. If any of the positions improves the quality of the layout, they move the current vertex to the position with the largest improvement among those positions where the topology of the layout does not change. They observe a typical problem of local optimization: overlong edges are often not shortened since this would need moving several vertices at the same time, a situation that corresponds to a local minimum of their target function. For a *bridge*, i.e., an edge whose removal disconnects the graph, this problem can easily be fixed by moving all nodes of the smaller component towards the larger component. After each iteration they detect all bridges and run the above fix for each of them. Of course the same problem can occur in doubly connected graphs, where the fix does not work. Stott and Rodgers have experimented with enforcing relative position, but report that the results were disappointing as there were many situations where a better layout could only be found by violating the relative position of some vertices. They label stations, but do not check for overlaps other than with the edges incident to the current station. They optionally use the same method for collapsing chains of degree-2 vertices as Hong et al. [HMN05] to preprocess the input graph.

Just recently, Stott and Rodgers presented a refined version of their method [SR05]. Improvements comprise mainly vertex labeling. They model labels as horizontal rectangles and consider eight different positions for each label. Labeling is performed after each iteration of vertex movements. The number of intersections between labels and either edges, vertices or other labels is included as a summand into the target function. Including labeling into the iterative layout process causes the layout to take into account the presence of labels. That is not the case when labels are placed as a second step once the layout is computed. However, in each iteration the two tasks are still separated. Obviously, their refined method cannot remove degree-2 vertices any more since they, too, need to be labeled.

The running time[1] of Stott and Rodger's method is considerably higher in comparison to Hong et al. [HMN05]. However, in the resulting maps nearly all edges are octilinear and edge lengths are quite uniform, which makes the maps more legible. The improved labeling method works nicely in some examples. Still, octilinearity of the result and overlap-free labeling is not guaranteed. Difficulties arise mostly because of the existence of local minima of the target function as the authors state themselves. The effects of local minima are manifold and some are described in [SR05]. Leaving a local minimum means in terms of the drawing that multiple vertices and labels would need to be moved simultaneously to further improve the quality measure.

The main advantage of our method over its predecessors is that we *guarantee* to draw octilinear metro maps, that we avoid the problem of local optima, and that we combine the placement of non-overlapping labels with graph drawing.

3.3 Graph Labeling

Two other approaches that use MIP formulations to combine graph drawing and label placement were described by Klau and Mutzel [KM99, KM03] and by Binucci et al. [BDLN05]. Unlike our MIP formulation, their approaches follow the topology-shape-metrics approach (see Section 5.1.1 or [dBETT99]) and are restricted to orthogonal drawings. Klau and Mutzel [KM99] gave the first algorithm that simultaneously draws a graph and labels its vertices. Given an orthogonal representation

[1] We give a more detailed comparison of results and running times in Section 7.4.

of a graph and vertex labels, their MIP formulation draws the graph orthogonally and attaches labels to its vertices while minimizing the total edge length. Binucci et al. [BDLN05] gave a MIP and several heuristics for drawing graphs orthogonally. In contrast to Klau and Mutzel, Binucci et al. allow edge labels, model vertices as rectangles that can contain textual information, and minimize the total drawing area.

Chapter 4

Drawing Metro Maps

Public transport maps are present in the daily life of many people today. Not only large cities use diagrams to visualize their public transport system but also smaller towns with just a couple of bus routes often display some sort of schematic diagram of these services. The common purpose is to provide the passengers with an easy-to-read map helping them to navigate through the transport network. In this chapter we define our notion of a metro map and summarize widely accepted design principles for "good" metro maps. Then we translate these information-visualization concepts into formal mathematical terms and define the computational problem of drawing a metro map.

4.1 What is a Metro Map?

Nowadays, people living or working in a large and developed metropolis often use its metro[1] system regularly. These commuters do of course have a strong picture of their own network map in mind when discussing the term *metro map*. This makes it difficult to come up with a proper definition. A recent book by Ovenden [Ove03] displays over one hundred different metro maps of the world. Although some common themes in the design of these maps reappear over and over again, there are many differences as well. It even seems as if countries developed their own metro map style over the years and there are subtle typical features for each of them.

What is common to all metro maps is that they depict the graph consisting of the metro stations as vertices and their interconnections as edges. We call this graph the *metro graph*. Thus, designing a metro map can be seen from both a cartographic and a graph-drawing perspective. Usually the metro graph is given as a geographic map of the city which includes the true location of all stations and connecting tracks. Only few cities stick to the geography in their metro map by simply emphasizing the metro lines overlayed in contrasting colors on the city map. Studying Ovenden's collection of metro maps we observed that the larger the metro graph the more cities use schematic maps.[2]

[1] The word 'metro' originates from the french *métro*, which is short for *chemin de fer métropolitain*. 'Metropolis' itself comes from Greek *meter* mother + *polis* city. Synonymous terms for metro are subway, underground, U-Bahn or just tube.

[2] An exemption is the map of the New York metro, with 469 stations one of the largest metros in the world, which returned from a schematic map in the 1970s to a geographical map today. However, some parts, where lines follow the grid layout of the roads, still have a rather schematic appearance.

The big advantage of schematic maps is that the cartographer is allowed to distort the scale and to straighten all the meandering lines of the geographic map. In a schematic map the topology is emphasized, not the location, as Morrison states in his survey on public transport maps [Mor96]. This freedom at hand, schematic metro maps can substantially reorganize a previously confusing network map. The mother of all modern schematic metro maps and a true design classic is the 1933 London Underground Diagram by Harry Beck, an engineering draughtsman. His map has an interesting history in its own [Gar94] and shows how well he managed to reorganize the London Underground or as Ovenden [Ove03] put it: "Beck's diagram resembled precision wiring rather than a plate of spaghetti!". Figure 4.1 shows a historic, geographic pocket map of 1920. In contrast, see Beck's first official schematic diagram of 1933 in Figure 4.2. Certainly, this revolutionary schematic map was the turning point in metro-map design and soon other cities came up with their own schematic maps, some well-done, others rather poor copies of Beck's map.

Schematic maps are most appropriate for underground railway systems or railways that are independent of the road traffic and only to a lesser extent for tram systems, where trams follow roads, or even for maps of bus routes. Morrison [Mor96] argues that usually passengers of transport systems on the surface, especially when following roads, have some clue about their true location. Then, it might be disturbing when looking at a schematic map where things are shown differently. However, on underground trains the journey is perceived as going straight and there is no disturbance since nothing in front of the windows needs to be matched with the metro map. Nonetheless, for many passengers schematic maps are helpful for route planning even in a surface-based transport network. Indeed, that is what a good metro map is all about: route planning. And the information needed to navigate in the network is exactly what is given in the metro map, namely the stations and their interconnections. When travelling with the metro one wants to know where one is, which station follows which on one's line, where one has to change to a different line, what are the names of the termini, and where to get off the train. A good schematic metro map aids in answering these questions (see [Mon96]) and is specifically designed such that the form follows this function.

With all this in mind, we now define our notion of a metro map as a schematic drawing of a public transport network. The minimum requirement for a schematic map is that only a limited number of line orientations may be used. The next section deals with different esthetics criteria which make up "nice" metro maps and which should be adopted by a method that sets out to automate the drawing of metro maps.

4.2 Metro-Map Esthetics

Esthetics criteria of metro maps can be divided into two groups: graphics criteria and layout criteria. The former deal, for example, with how stations and lines are graphically represented or which font to use for labeling. This is beyond a graph drawing algorithm and must be decided by the user. The latter concern the structure of the drawing, i.e. where to put stations and interconnections on the map. This is also what needs to be done in an algorithm for drawing metro maps.

4.2.1 Graphical Criteria

Graphics criteria, such as the colors of the lines, strongly influence the readability of a metro map. However, they are all based on empirical studies and the experience

of professional graphic designers. See for example Tufte's book on information visualization [Tuf90]. The most important principles for metro maps include:

- Code the different transport lines in the system by contrasting colors.[3] This is important for not getting confused when two lines meet at an interchange station. It must be clear where each line continues.
- Emphasize the names of the termini of all lines such that passengers can quickly determine the right direction of each line.
- Use different station symbols for interchange stations and regular stations. Stations are usually marked with circles, boxes or just ticks but this varies between different maps. Emphasize interchange stations.
- Label the stations using as few text orientations as possible. Labels should of course not overlap with other elements of the map. Use a clear font.

This list is certainly not complete but it contains the principles that we included in our graphical output (see Chapter 7). We use the same color coding as in the corresponding official maps. Station symbols are similar to the London style, i.e. interchange stations use black circles with white centers and regular stations are just tick marks.

4.2.2 Layout Criteria

Identifying the right layout criteria is important for modeling the metro map problem in the next sections. Beck's revolutionary diagram from 1933 (see Figure 4.2) already satisfies most of the criteria by which contemporary metro maps are designed. The following list gives the principles which we found in most of the maps in Ovenden's book [Ove03].

- Restrict all line segments to the octilinear orientations. This is the most common layout style used today, although in some instances diagonals at 60° are utilized. Compare for example the Madrid metro map in Figure 4.3, where diagonals at 60° help to decrease the height of the map, with the classic Beck diagram in Figure 4.2 and its diagonals at 45°.
- Roughly support the geographical mental map of passengers. This means that the embedding of the graph must not be changed and that the relative position between stations is roughly kept. A station which is located north of some other station should not be placed south of that very station in the metro map. Also, if some important geographical feature such as a coastline or river is present, this feature might be included on the map (as the river Thames on the tube map, see Figure 4.2) to help the user maintain his mental map.
- It is important that lines are easy to follow with the eye. Hence, there should be as few bends as possible along the lines. If they are unavoidable, then they should be as soft as possible, i.e. preferably using an angle of 135°. Special care needs to be taken of stations where two or more lines meet. Lines should always continue straight on both sides of that station, otherwise the eye might be misled and continue on the wrong line.

[3] Maps using color codes for different lines are in "French style" according to Morrison [Mor96].

- For a balanced look, the distances between adjacent stations should be as uniform as possible. On monotonous parts between two interchanges all stations should be equally spaced.
- Leave enough space for the station labels in the drawing. Ideally, the space requirements for station labels are considered *during* the layout process.

Similar layout criteria for metro maps have been identified by both Hong et al. [HMN05] and Stott and Rodgers [SR04]. Clearly, each metro map is a compromise between the above principles. For example a small number of bends often means that the geography needs to be quite heavily distorted. Therefore, the difficulty lies in finding the right balance between all of these criteria.

4.3 Modeling Metro Maps

Given the above design principles we define a set of hard and soft constraints for "nice" (unlabeled) metro maps. The hard constraints are compulsory and must be fulfilled by each metro map layout. They implicitly define the space of all admissible metro maps. The soft constraints, however, should hold as tightly as possible and are used to measure the quality of a metro map. The better a map conforms to the soft criteria the better its quality.

So, let $G = (V, E)$ be the plane input metro graph and let the line cover $\mathcal{L}$ contain all metro lines as paths and cycles. If the input is not a plane graph because for example two metro lines intersect without having a station at the intersection, we can simply planarize the graph by introducing a dummy vertex[4] at this intersection. Further, if there are metro lines that contain branches, and hence do not form a path or cycle, we can treat each branch as a metro line on its own. So we can safely assume that G is a plane graph and $\mathcal{L}$ consists of paths and cycles.

4.3.1 Hard Constraints

With the hard constraints we model the crucial parts of a metro map. Which characteristics must be present in any case? In our opinion this is above all octilinearity and preservation of the embedding among some minor, but nonetheless important, properties. The exact constraints are as follows.

(H1) The drawing must respect the input topology, i.e. the embedding of the plane input graph G must be preserved,

(H2) all edges in the drawing must be straight, octilinear line segments,

(H3) each edge e has a minimum length ℓ_e, and

(H4) each edge has minimum distance $d_{\min}$ from each non-incident edge.

Constraints (H1) and (H2) enforce topology-preserving, octilinear straight-line drawings. (H3) and (H4) prevent that two vertices can get arbitrarily close in the drawing. For two adjacent vertices their minimum distance is given by (H3). Usually the minimum distance is the same for all edges. Constraint (H4) not only sets a

[4] In our implementation these vertices are treated slightly different. For example lines are not allowed to bend at dummy vertices.

minimum distance for non-adjacent vertices but also prevents edge crossings and thus enforces planarity of the drawing. While generating planar topology-preserving metro-map drawings has been done before [HMN05, SR04], using octilinearity as a compulsory criterion is new. Octilinearity is highly important in real-world metro maps. Once a cartographer has decided to design an octilinear map, he will not tolerate a single non-octilinear line because it would disrupt the overall appearance of the map.

4.3.2 Soft Constraints

Having set the mandatory requirements for a metro map we can now give the soft constraints that allow to distinguish between good and poor metro maps. The soft constraints model the remaining layout criteria as follows.

(S1) Each metro line in $\mathcal{L}$ should have few line bends, especially at interchange stations,

(S2) the total length of all edges in the drawing should be small, and

(S3) for each pair of adjacent vertices their relative position should be respected as well as possible.

Constraint (S1) directly models the bend criterion. This is a very important criterion for clear octilinear layouts. Constraint (S2) must be seen in conjunction with (H3). These two constraints model the ideal edge length in the drawing. Since (H3) sets a minimum edge length and (S2) minimizes the total edge length, the optimal edge length for each edge exactly equals its minimum length. Constraint (S3), finally, means that the layout should not diverge more than necessary from the input layout. If possible, each edge should have the octilinear direction which is closest to the original direction. By measuring the angle between the geographic and schematic direction of each edge this criterion can be quantified.

4.4 The Metro-Map Layout Problem

Now that we have given hard constraints that need to be satisfied by a metro map and soft constraints that should be optimized, we can formally state the metro-map layout problem. For now we only consider the pure graph-drawing task without labeling.

Problem 4.1 (Metro-Map Layout Problem) Given a plane graph $G = (V, E)$ with maximum degree 8 and vertex coordinates in $\mathbb{R}^2$, a line cover $\mathcal{L}$ of G, minimum edge lengths $\ell_e > 0$ for each $e \in E$ and a minimum distance $d_{\min} > 0$,
find a *nice* drawing Γ of G, i.e. a drawing Γ that satisfies the hard constraints (H1)–(H4) and optimally fulfills the soft constraints (S1)–(S3).

First, note that the restriction to graphs with maximum vertex degree 8 is justified since octilinear drawings can by definition only be realized for graphs with maximum degree 8. Second, the user must define the actual objective function by setting the weights of the three constraints (S1)–(S3) according to his preferences. Clearly, different weights result in different drawings. Further note that the objective function can be easily extended by including other soft constraints.

If we combine graph drawing and labeling, the only difference to Problem 4.1 is that we have additional hard constraints that model non-overlapping labels which are placed according to one out of a set of predefined label positions. We show in Section 6.4 how to extend our model to solve the graph-labeling problem.

Before we show how the metro-map layout problem can be formulated as a mixed-integer program and how it can be solved in practice, the next chapter will analyse the computational complexity of the problem.

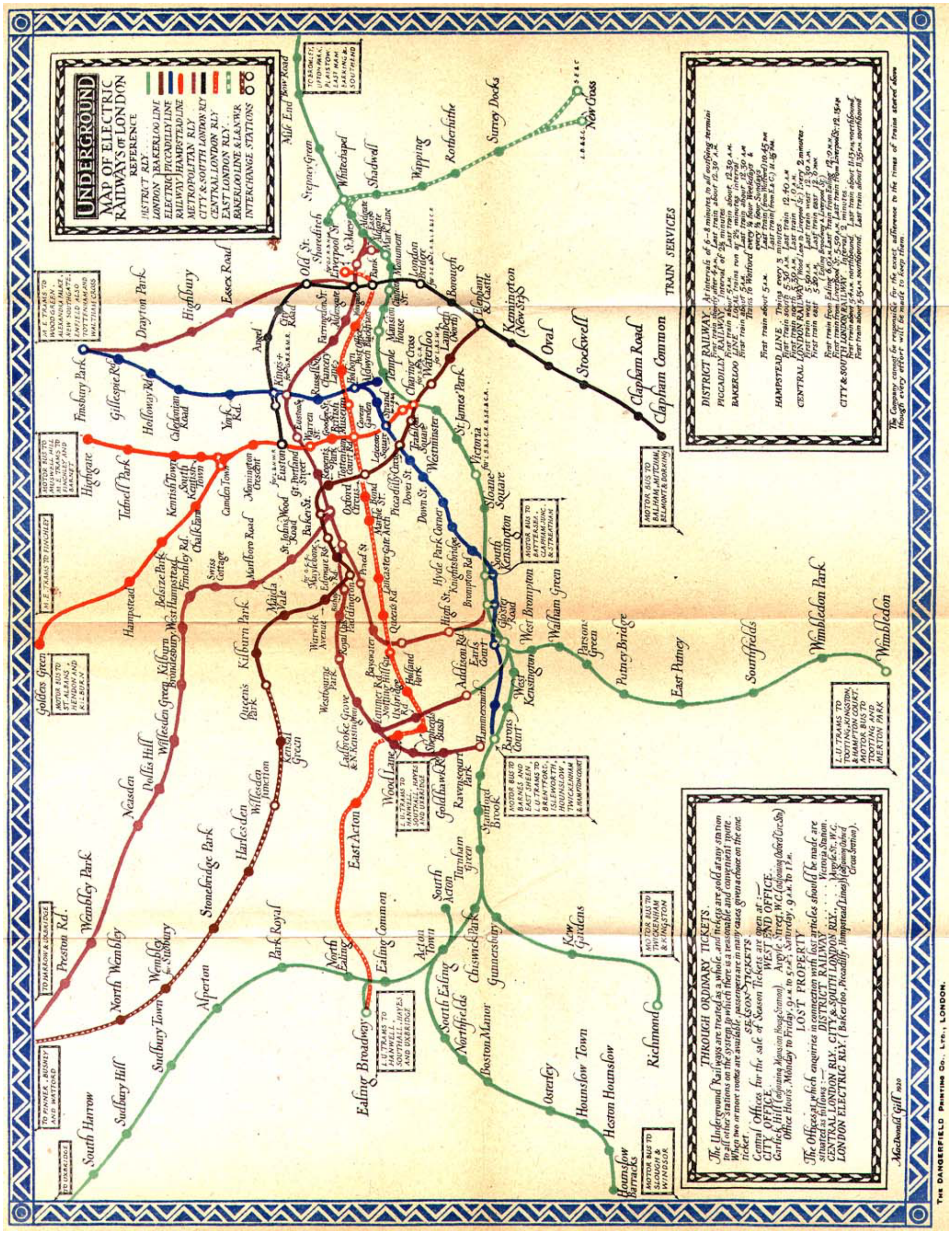


Figure 4.1: A geographic tube map of 1920 [ltm] by MacDonald Gill.

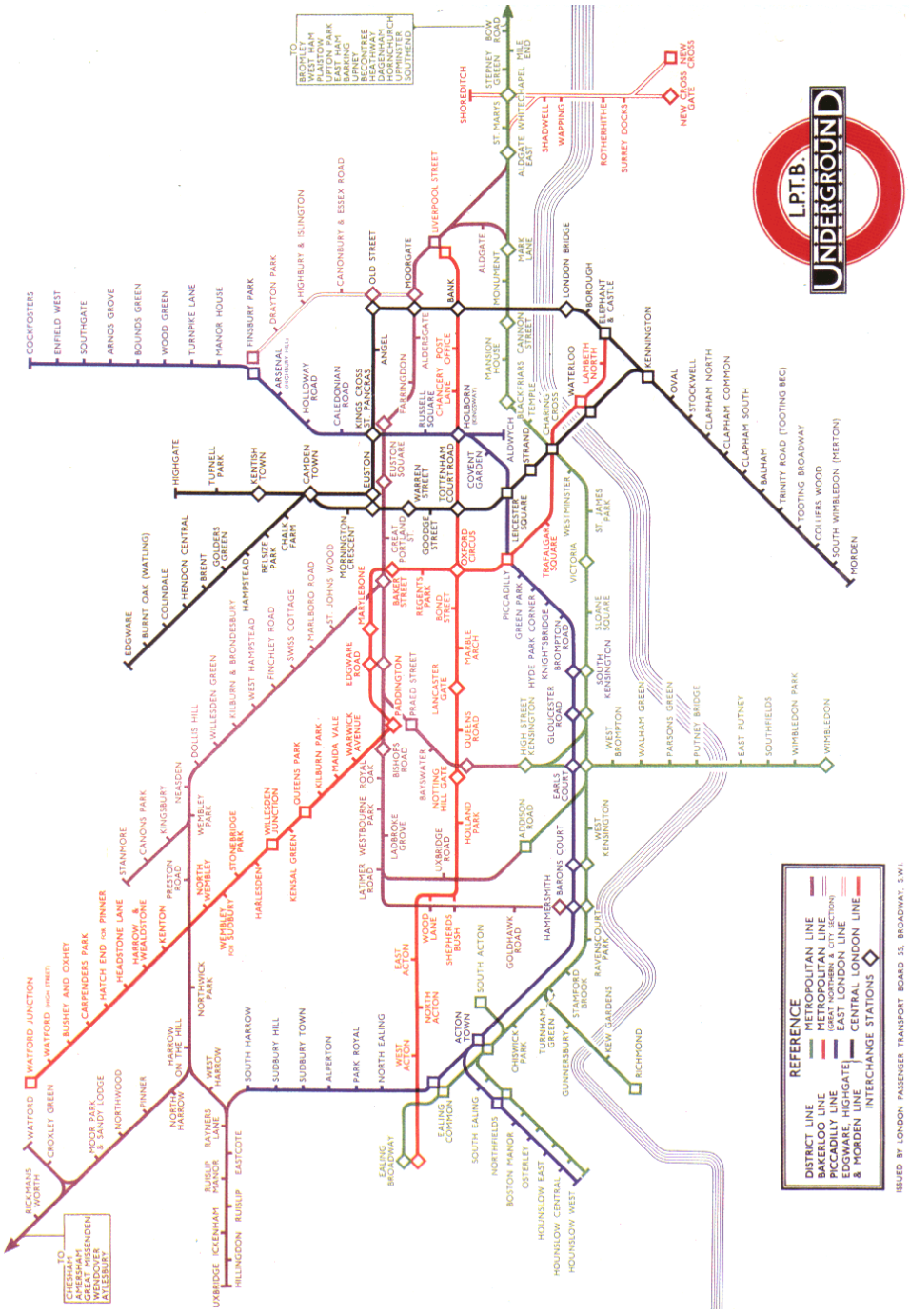


Figure 4.2: Harry Beck's first official tube map of 1933 [ltm].

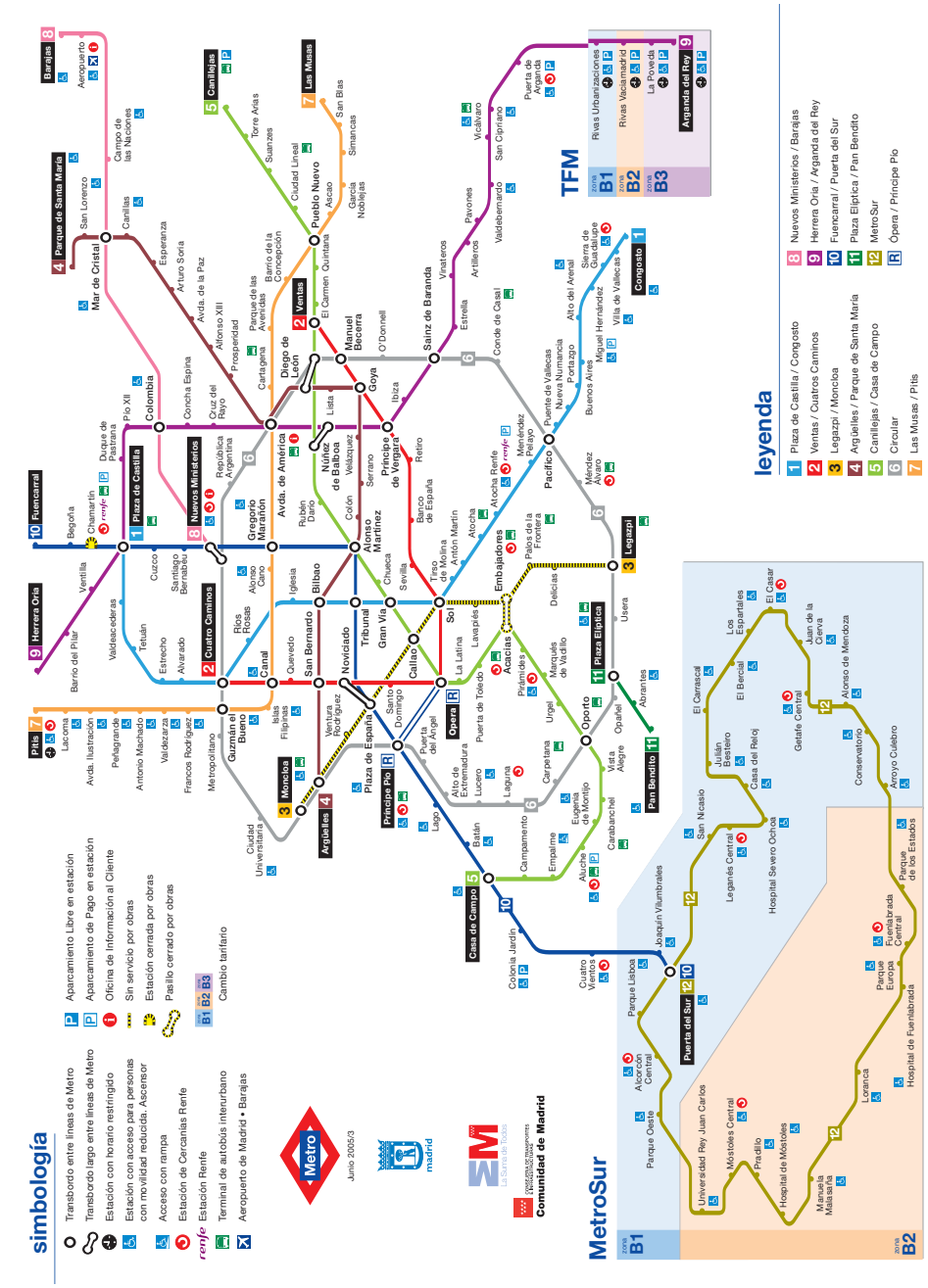


Figure 4.3: The Madrid metro map [Met] is an example of a modern metro map using diagonals at 60°.

Chapter 5

Complexity of Drawing Metro Maps

In this chapter we give theoretical results about the complexity of drawing metro maps. Section 5.1 introduces the seemingly similar and well-studied problem of drawing orthogonal layouts and shows that previous complexity results for orthogonal layouts cannot be carried over to octilinear layouts like metro maps. Therefore, in Section 5.2, we show the NP-completeness of the metro-map decision problem, which is in contrast to the existence of efficient decision algorithms for the orthogonal case.

5.1 Relationship to Orthogonal Layouts

Orthogonal or rectilinear graph drawing methods (see [EFK01] or [dBETT99] for an overview) were first developed in the context of VLSI chip design but soon these layouts have also been studied for many other applications such as entity-relationship or data-flow diagrams for databases or in software engineering. Orthogonal layouts have a very schematic appearance since all edges are drawn as axis-parallel paths. The difference to octilinear layouts is that the latter allow two more orientations in the drawing, namely the two diagonals. In spite of this rather small change in the definition, the existence of straight-line drawings in the rectilinear setting can be efficiently answered, whereas it turns out to be NP-hard for octilinear layouts. Next, we show how to answer the existence of orthogonal layouts with metro-map-like properties using Tamassia's *topology-shape-metrics* algorithm [Tam87]. Then, we outline the difficulties of the topology-shape-metrics method for octilinear layouts.

5.1.1 Rectilinear Graph Drawing

If we were to draw orthogonal layouts with properties similar to metro maps then the corresponding decision problem is solved efficiently as stated by the following theorem.

Theorem 5.1 (Tamassia 1987) *Let $G = (V, E)$ be a plane graph with maximum degree 4. Then there is an efficient algorithm to decide whether G can be drawn as an orthogonal metro map, in the sense that*

1. *all edges are drawn as rectilinear line segments,*
2. *the embedding of G is preserved, and*
3. *the layout is planar.*

Proof. Tamassia [Tam87] presented an algorithm that computes a bend-minimal orthogonal layout for a given plane graph G with maximum degree 4. As required above the resulting orthogonal layout is planar and preserves the embedding. The algorithm transforms the layout problem into a network flow problem, more precisely it involves finding a minimum-cost flow in a network with $O(n)$ vertices and arcs, where $n = |V|$. Currently, the best algorithm [GT96] for this network-flow problem requires $O(n^{7/4}\sqrt{\log n})$ time. The cost of a flow in this network corresponds to the number of bends in a drawing of G. Hence, a drawing that satisfies the above properties exists if and only if the minimum-cost flow in Tamassia's algorithm has cost equal to 0. □

Tamassia's seminal work [Tam87] established the class of *topology-shape-metrics* algorithms. In the topology-shape-metrics approach a graph layout is computed in three steps. The first step finds a planar *topology* for the input graph G. Since in our case we are already given an embedded graph, this step can be omitted. The second step determines the *shape* of the layout. This is done by finding an *orthogonal representation* of G. Basically, this is a list of face descriptions, where each face description consists of a list of the circularly ordered edges on its boundary. For each of these edges its bends are described and the angle between its last segment and the first segment of the next edge in the face description is given. There is a set of four conditions for which Tamassia shows that they are necessary and sufficient for an orthogonal representation to correspond to a planar orthogonal layout. The conditions model that the sum of all angles at a vertex equals 360°, that the sum of all angles inside an internal face f equals $(180 \deg(f) - 360)^\circ$ and $(180 \deg(f) + 360)^\circ$ on the external face, that the bends on an edge e are compatible in both face descriptions in which e appears, and, finally, that there is a planar graph that belongs to this orthogonal representation. We call an orthogonal representation satisfying these conditions *feasible*. Note that the bend minimization used in the proof above belongs to this second step and finds a feasible orthogonal representation with a minimum number of bends. Finally, the third step, known as *compaction*, sets the actual coordinates of all vertices and bends, thus fixing the *metrics* of the layout. Usually some criterion like total edge length or area of the drawing is optimized.

5.1.2 Extension to Octilinear Layouts

The flow network in the bend-minimization step of Tamassia's algorithm can be extended to so-called *k-gonal* graphs [Tam87]. A k-gonal graph is a plane graph where the angles between edge segments are integer multiples of $180/k$ degrees. For $k = 2$ this is just the orthogonal situation from the previous section but for $k = 4$ we get octilinear layouts. Unfortunately, in contrast to orthogonal representations, a feasible k-gonal representation is not sufficient for the existence of a suitable layout. This observation is depicted in Figure 5.1. It shows a planar graph which is drawn octilinearly, but with two edge intersections. These intersections are unavoidable when drawing the graph with the given angles. However, the underlying 4-gonal representation is feasible. The shape of the thin lines in the figure is fixed (except scale) if the given angles must be realized. Only the thicker path can be modified as

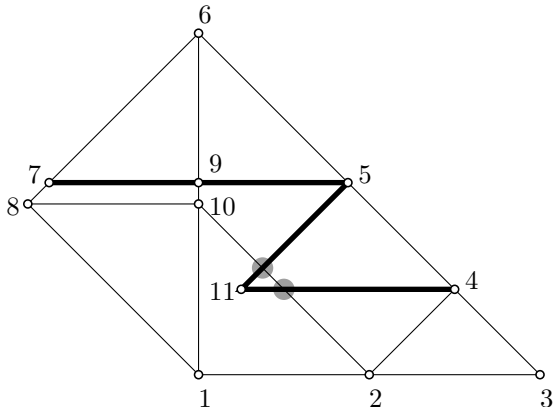


Figure 5.1: A 4-gonal graph which cannot be drawn planarly although its 4-gonal representation is feasible. The two edge intersections marked by gray circles are inevitable regardless of changing edge lengths or vertex coordinates.

a whole by shifting its upper part up or down. But because the lower right vertex of the path is anchored at a fixed position there will always be at least two edge intersections as shown in the figure, regardless of how the upper part of the path is shifted.

A similar but more general observation has been made by Bodlaender and Tel [BT04]. They use the notion of d-linearity for graphs that allow a planar drawing with all angles integer multiples of $360/d$ degrees. Note that a k-gonal graph as defined in the previous paragraph is identical to a $2k$-linear graph in Bodlaender and Tel's terminology. For $d > 4$ they show that an angular resolution of $360/d$ degrees does not imply d-linearity. This leads to the conclusion that for $d > 4$ a feasible d-linear representation for a plane graph G, as determined by Tamassia's flow network, does not suffice to allow a d-linear drawing of the graph. The final compaction step in the topology-shape-metrics approach fails for certain inputs. Hence, the existence of a feasible octilinear or 4-gonal representation is not equivalent to the existence of a corresponding octilinear layout. In the next section we show that the latter existence problem is NP-hard.

5.2 NP-Completeness of MetroMap

In this section we formulate a decision version of the metro map layout problem and subsequently show that this problem is NP-complete. The proof is by reduction from the NP-complete problem Planar 3-Sat, which is described next.

5.2.1 Planar 3-Sat

Recall the 3-Sat problem. Given a Boolean formula φ in conjunctive normal form, i.e. $\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where each clause c_i contains exactly three literals (a literal is a variable or its negation), the problem is to decide whether there is an assignment of values to the variables that satisfies all clauses. The 3-Sat problem is known to be NP-complete.

The Planar 3-Sat problem [Lic82] is a satisfiability problem just as regular 3-Sat with the difference that the Boolean formula whose satisfiability is tested has to be planar, which is defined in the following.

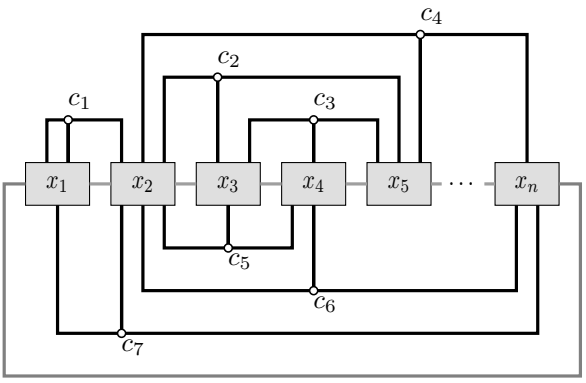


Figure 5.2: Example of a planar variable-clause graph. The variables are placed on a horizontal line and the clauses are represented as non-intersecting three-legged combs connected to their variables either from above or from below. The inter-variable edges shaded in gray are often left out in such a drawing.

Definition 5.1 *Let φ be a Boolean formula in conjunctive normal form. Let $X_\varphi = \{x_1, x_2, \ldots, x_n\}$ denote the set of variables of φ and $C_\varphi = \{c_1, c_2, \ldots, c_m\}$ the set of clauses of φ. Then $H_\varphi = (V, E)$ is called the* variable-clause graph *of φ, where*

$$V = X_\varphi \cup C_\varphi$$

and

$$E = \{x_i c_j \mid x_i \in c_j \text{ or } \overline{x_i} \in c_j\} \cup \{x_i x_{i+1} \mid 1 \leq i < n\} \cup \{x_n x_1\}.$$

Now, we can introduce the notion of a planar formula using the planarity of its variable-clause graph.

Definition 5.2 *A Boolean formula φ in conjunctive normal form with at most three literals per clause (3-CNF) is called* planar *when its variable-clause graph H_φ is planar.*

In other words this means that the variable-clause graph of a planar formula can be drawn such that all variables are placed next to each other on a horizontal line and the clauses are drawn as three-legged combs connected to those variables that occur in that clause [KR92]. These clause-combs lie either completely above the variables or completely below. See Figure 5.2.

In 1982 Lichtenstein [Lic82] proved the following theorem about Planar 3-Sat.

Theorem 5.2 (Lichtenstein 1982) *Deciding the satisfiability of a planar 3-CNF formula is NP-complete.*

This result has been used before in several NP-completeness proofs for geometric problems such as point labelling [SW01] or graph drawing [CDR03].

5.2.2 MetroMap is NP-Complete

Having introduced Planar 3-Sat we can now show that MetroMap is an NP-complete problem as well. We formulate the following theorem about the existence of an octilinear layout with metro-map properties analogous to Theorem 5.1.

Theorem 5.3 *Let $G = (V, E)$ be a plane graph with maximum degree 8. Then it is NP-hard to decide whether G can be drawn as a metro map, in the sense that*

1. *all edges are drawn as straight, octilinear line segments,*
2. *the embedding of G is preserved, and*
3. *the layout is planar.*

Proof. As stated before, we prove this result by reducing Planar 3-Sat to MetroMap. Hence, we have to find a polynomial transformation which maps a planar 3-CNF formula φ to a plane graph $G(\varphi)$ such that

$$\varphi \text{ satisfiable} \Leftrightarrow G(\varphi) \text{ metro-map drawable.}$$

Instead of considering φ itself, we think of the planar embedded variable-clause graph H_φ as the object that will be transformed. Indeed, we construct the graph $G(\varphi)$ in a way that its overall structure resembles H_φ. One type of graph substructures or *gadgets* will model the variables, i.e. these gadgets can be drawn in exactly two conformations representing the truth assignments of the respective variable. The second type of gadgets will represent the clauses of φ, so it has the shape of the combs in H_φ (recall Figure 5.2) and is able to transmit the truth values of the literals involved. At the point where the three legs meet there is a structure that admits a planar drawing if and only if at least one of the literals has the value *true*. Thus, we can draw $G(\varphi)$ as a metro map if and only if φ is satisfiable.

The construction of $G(\varphi)$ uses three basic building blocks that are depicted in Figures 5.3 to 5.5. The main observation is that in the octilinear setting non-degenerate triangles can only have three different shapes as shown in Figure 5.3(a). Since in a triangle the sum of all three internal angles must equal 180° and since we do not allow degenerate angles of 0°, each angle must be of at least 45°. Hence, it follows immediately that exactly one of the three angles is 90° and both remaining angles are 45°. So the degree of freedom when drawing an octilinear triangle with one fixed side consists in the choice of the vertex to be adjacent to the 90° angle.

Now we can combine several triangles to form more complex structures such as the square block in Figure 5.3(b). We now argue that this block can only be realized as depicted in the figure. The underlying embedded graph with five vertices has four triangular faces ABE, BCE, CDE and DAE that share vertex E. Further, E does not belong to any other face. Hence, the only way of assigning the right angles in the four triangular faces is to put them at vertex E, otherwise the angles at E would not sum up to 360°. This already fixes all angles in this block structure. Obviously, larger rigid structures can be built by attaching these square blocks side-by-side.

The third building block will model a translational joint between two rigid components. In a first step we connect two square blocks by a construction consisting of two triangles as in Figure 5.4. Because each of the two triangle has three possible conformations we get in total nine different octilinear realizations of this component. In order to serve as a joint with only two positions we need to find a way to restrict the number of realizations to just two. The first two columns in Figure 5.4 show drawings of the component where the two square blocks involved are not parallel or of different size. Only the last column shows three configurations that move one square block with regard to the other one such that they remain parallel and equally sized. We first modify the joint by placing another copy next to it as depicted in Figure 5.5(a). Note that by combining the two copies like this the unwanted configurations mentioned above are ruled out. It remains to get rid of

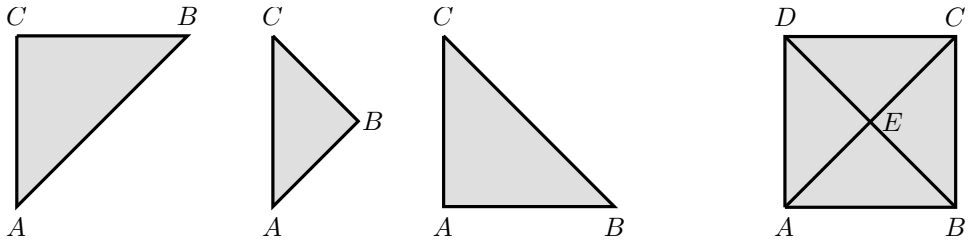


Figure 5.3: Two basic building blocks for the gadgets. A triangle with fixed side AC can be realized octilinearly in exactly three different ways as shown in (a). The square block in (b) has a fixed octilinear shape.

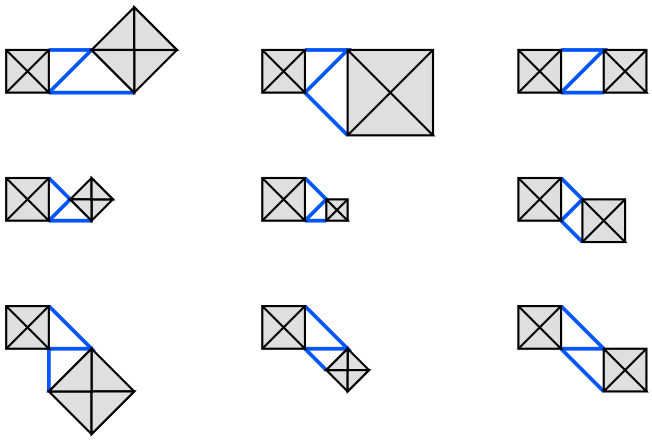


Figure 5.4: The nine possibilities of drawing two square blocks connected by two octilinear triangles that share one side.

the realization on the right-hand side of Figure 5.5(a) that not only moves the two bars vertically but also decreases their distance. To that end we attach a block with a triangle and two opposed blocks to the bars as depicted in Figure 5.5(b). The triangle serves as a spacer to keep both bars at the desired distance. In the first two configurations of Figure 5.5(b) the spacer does not interfere with the opposed bar but the last configuration cannot be correctly drawn without violating planarity. So finally, this construction indeed models a translational joint that can shift a bar by exactly one step in terms of the square block size. Further, the distance between both bars equals the square block size, too.

Now we can construct more complex structures that serve as gadgets for the variables and the clauses. It is important that all parts of these gadgets are connected such that the side lengths of the square blocks are equal. This is ensured when connecting square blocks side-by-side or when using the translational joint. In that case we can assume that all vertices are placed on a uniform grid with unit length and we do not have to deal with differently scaled substructures.

First, we describe the variable gadget. It must have the property to be drawable in exactly two configurations that represent the truth values of the corresponding variable. Further, it must be able to transmit that truth value to the clauses containing the variable, depending on whether it appears as negated literal in the respective clause or not. A sample variable gadget is shown in Figure 5.6 and explained in the following.

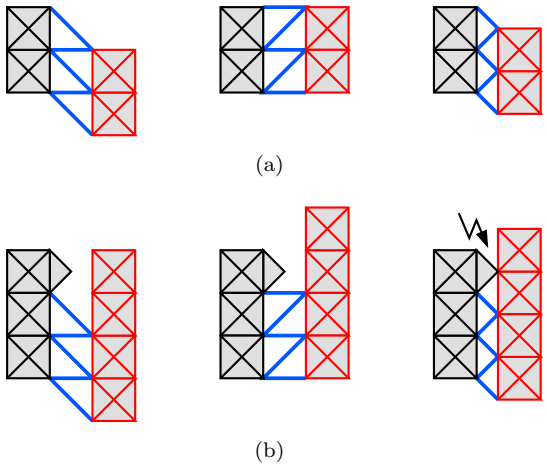


Figure 5.5: Modified octilinear structure that models a translational joint. In subfigure (a) the structure has three possible planar conformations. Including three more blocks and an additional triangle as a spacer in subfigure (b), only two planar conformations remain.

The main part of this gadget is a large horizontal bar (*variable bar*) in the middle of the construction made up of of square blocks and containing some dents. It is framed by a box of square blocks with upward and downward ports that will later be connected to the clause gadgets. The bar can take a position on the left-hand side (Figure 5.6(a)) and a position on the right-hand side (Figure 5.6(b)) since it is fixed to its frame by several translational joints. Let the left position represent the variable's value *true* and the right position *false*. The connections to the clauses are shown as the openings of *tunnels* with vertical *literal bars* inside. These literal bars are fixed to the sides of the tunnel by translational joints and hence can move upwards and downwards. They are placed relative to the variable bar such that they can only be moved towards it if there is a dent aligned with the tunnel opening. If the dent is shifted away from its aligned position then the vertical bar in turn must be shifted away from the variable bar and into its tunnel, otherwise planarity would be violated by the touching bars. Thus, the literal bar transmits pressure into the clause gadget. The placement of the dents on the variable bar is as follows. Assume that the bar takes the position on the left-hand side. Then, for all connections representing positive literals we place an aligned dent on the bar. For negated literals the dents are placed one position to the left such that they align with the tunnel openings when the variable bar takes its right-hand position. Consequently, only those literal bars corresponding to literals evaluating to *true*, depending on the current value of the variable, can be fitted into their dents. This fact will be used for constructing the clause gadget. Note that *all* parts of the structure are connected such that they all use the same unit square block size.

Second, we describe the clause gadget. It must be constructed to be planarly realizable if and only if at least one literal of that clause is *true*, i.e. one of the literal bars in the tunnels connected to the variables fits into its dent. In other words the gadget may not be drawn correctly if there is pressure from all three literals bars. Now, we describe the clause gadget, as shown in Figure 5.7, in detail.

The gadget has the shape of a three-legged comb just as in the variable-clause graph of the Planar 3-Sat problem, see Figure 5.2. The tunnel corresponding to the second literal can be connected directly to its variable gadget. The two outer tunnels are making a turn to run horizontally towards the center. At this turn the

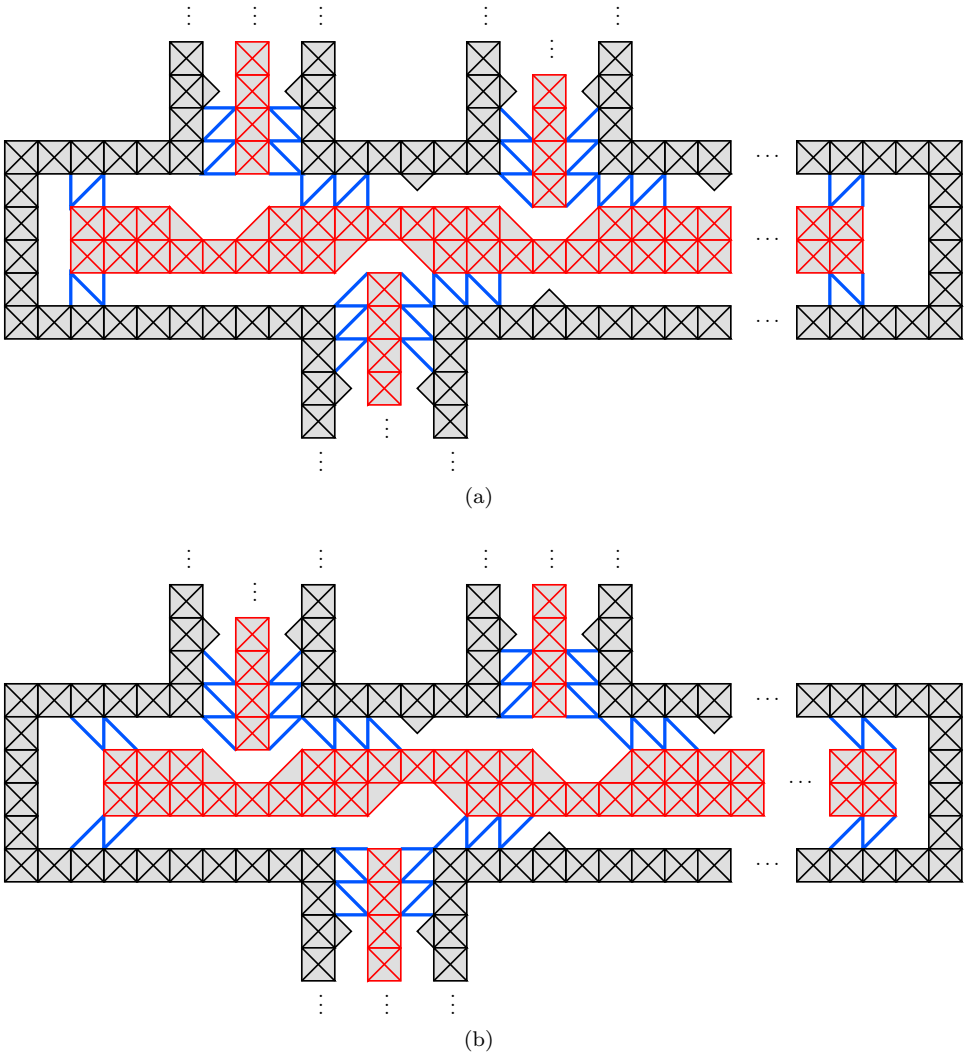


Figure 5.6: The variable gadget. Subfigure (a) shows the state corresponding to *true*, and subfigure (b) corresponds to *false*.

vertical pressure from the variable is transformed into horizontal pressure. This can be seen in Figure 5.7(a). The left literal bar is in the upper position and thus causes the adjacent horizontal bar to be in its right position. The literal bar on the right-hand side is in the lower position and therefore allows the corresponding horizontal bar to be shifted away from the center of the clause gadget. However, the crucial part of the clause gadget is a flexible switch in the center of the structure where the three literal tunnels meet. As a whole, this switch can shift vertically by one unit due to four joints fixing it to the vertical walls of the gadget. In Figure 5.7(a) it is in the lower position and in Figures 5.7(b) and 5.7(c) it is shifted upwards. The middle part of the switch consists of a bar made up of four square blocks and two triangles. This bar can be independently moved to the left and to the right using two standard translational joints with spacers. Note the three black triangles at the left, bottom and right side of the switch. These triangles may overlap with the triangles at the end of those literal bars that exert pressure into the clause gadget as seen in Figure 5.7(c) where the left literal bar and the switch bar touch each other and thus violate planarity at the flash symbol. All in all, the whole

gadget can be drawn as a metro map as long as at least one literal is *true* and the switch is shifted towards this very literal. However, if all literals are *false* and hence the respective bars are shifted inside the gadget, then all possible positions for the switch result in a violation of planarity. Again, all side lengths of the square blocks in the clause gadget are equal because all the different parts are connected via translational joints.

With these two gadgets for variables and clauses we can construct the whole graph $G(\varphi)$ by connecting the literal tunnels of the clause gadgets to their ports in the respective variable gadgets. The resulting graph is planar since φ is planar. We choose its input embedding according to a planar drawing of the variable-clause graph of φ and the above gadget structures.

To conclude, let us repeat the correspondence between a planar Boolean 3-CNF formula φ and the graph $G(\varphi)$ constructed above. If φ is satisfiable then $G(\varphi)$ can be drawn as a metro map, where each variable gadget is realized corresponding to the truth value in a satisfying variable assignment of φ. Consequently, by construction, each clause gadget is correctly drawable as well. If, however, φ is not satisfiable then for each variable assignment there is at least one clause that evaluates to *false*. This means that $G(\varphi)$ is not drawable as a metro map because in the corresponding clause gadget all literal bars are pushed towards the clause center so that none of the conformations of the switch can be drawn planarly.

Finally, the reduction itself can be done in polynomial time because $G(\varphi)$ is embedded on a grid of size polynomial in the length of φ. Therefore $G(\varphi)$ has a polynomial number of vertices and edges. □

Corollary 5.1 MetroMap *is NP-complete.*

Proof. Obviously, for a given drawing of a plane input graph G, one can check in polynomial time whether all edges are octilinear, the embedding is preserved, and the drawing is planar. Hence MetroMap belongs to the class NP. By Theorem 5.3 MetroMap is NP-hard. This implies the NP-completeness of the problem. □

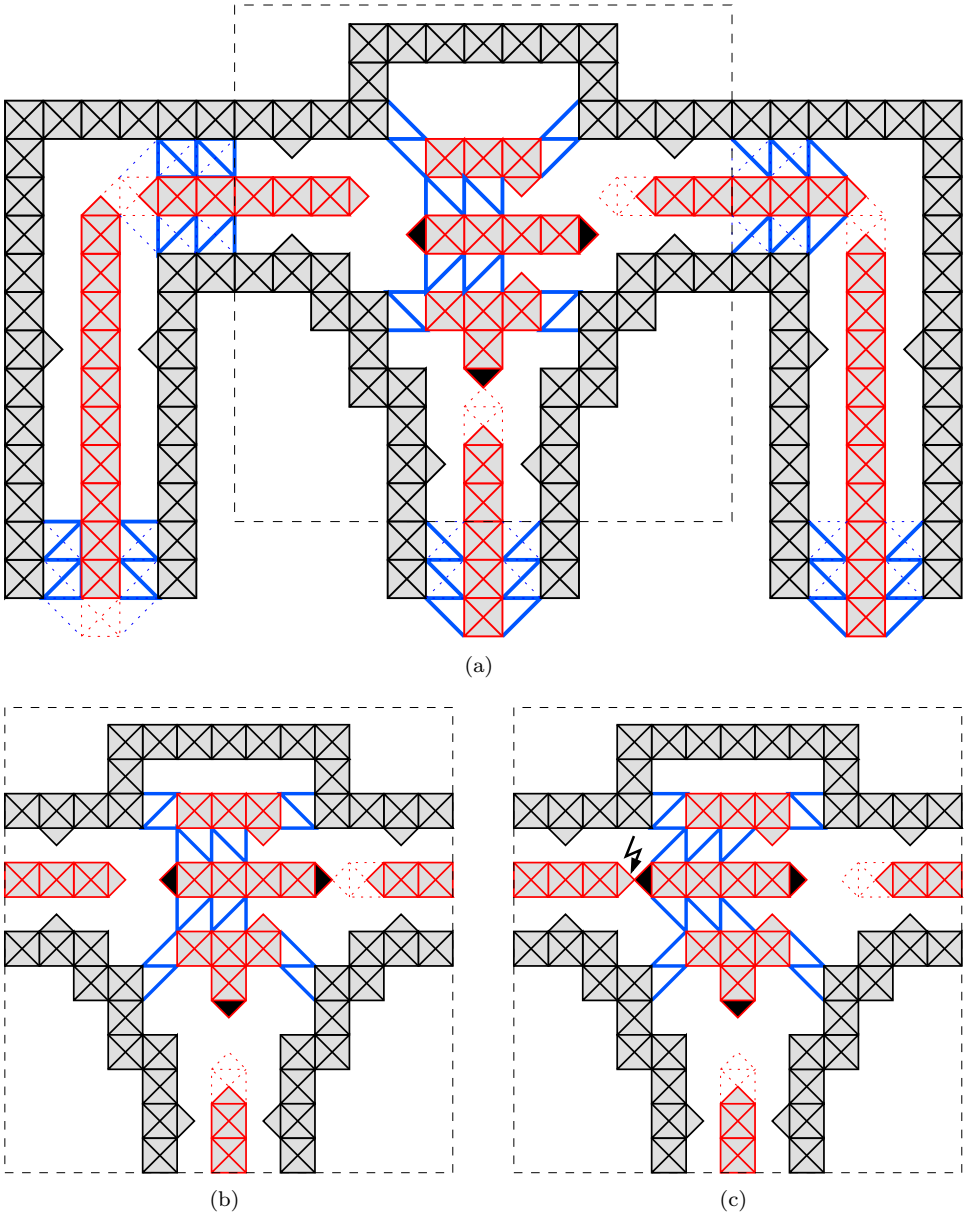


Figure 5.7: Three realizations of the clause gadgets. In subfigure (a) the literal bar on the left transmits the value *false*, the other two literal bars are in their lower position and transmit *true*. Alternative positions for these literal bars are indicated with dotted lines. The switch in the center of the gadget selects the second literal in subfigure (a). Subfigures (b) and (c) only show the central part of the gadget (indicated by dashed lines) containing the switch. This switch selects the third literal in (b), and the first literal in (c). Since the first literal is *false*, planarity is violated at the flash symbol in (c).

Chapter 6

A Mixed-Integer Program for Metro-Map Layouts

We decided to formulate the metro-map layout problem as mixed-integer program. As we have shown in Section 5.2, the metro-map layout problem is NP-complete which is a valid reason to apply a likewise NP-complete optimization method such as MIP. Using MIP gives us the necessary flexibility to achieve the following. If a layout that conforms to all hard constraints exists (and this was the case in all examples we tried), then our MIP finds such a layout. Moreover our MIP optimizes the weighted sum of cost functions each of which corresponds to a soft constraint.

As described in Section 2.2 a MIP consists of two parts: a set of linear constraints and a linear objective function. In Section 6.1 we give the linear constraints that model the hard constraints (H1)–(H4) of a metro-map layout. We simultaneously optimize the three soft constraints (S1)–(S3) in Section 6.2 using a weighted sum of three individual objective functions. The total number of constraints and variables in our model is $O(n + m' + m^2)$, where n is the number of stations in the metro graph, m is the number of edges, and m' is the size of the line cover $\mathcal{L}$. Note that, since G is planar, we have $m \leq 3n - 6$ due to Euler's formula. Section 6.3 gives heuristic methods to reduce the metro graph and the MIP size. In Section 6.4 we describe how vertex labels are included in the MIP model. Finally, Section 6.5 gives an overview of our prototype implementation.

6.1 Linear Constraints

The linear constraints of our MIP are grouped into four parts. The first part in Section 6.1.1 establishes an auxiliary coordinate system that eases handling octilinearity. The second part in Section 6.1.2 models *octilinearity*, *minimum edge length*, and even parts of the soft constraint *relative position*. Section 6.1.3 is the third part and describes the constraints that are necessary for the *preservation of the embedding*. The fourth part, the constraints in Section 6.1.4, ensures that edges do not intersect and keep their given *minimum distance*. Additional constraints that are necessary for the objective function will be described later in Section 6.2.

6.1.1 Coordinate system

To be able to treat all four edge directions similarly, we introduce an (x, y, z_1, z_2)-coordinate system, where each axis corresponds to one of the four admissible edge orientations in the layout (see Figure 6.1). Thus, each vertex v has not only an x-coordinate $x(v)$ and a y-coordinate $y(v)$ but also a z_1- and a z_2-coordinate. The latter are defined and modeled as constraints as follows:

$$\begin{array}{rcll} z_1(v) & = & x(v) + y(v) & \\ z_2(v) & = & x(v) - y(v) & \end{array} \qquad \forall v \in V, \tag{6.1}$$

where the variables $x(v), y(v), z_1(v), z_2(v)$ can be chosen as either integers or continuous and $0 \leq x(v), y(v) \leq N$. The constant N is meant to restrict the drawing area in a reasonable way. In our experiments (see Chapter 7) we chose $N = n$. This part of the model uses $2n$ constraints and $4n$ variables.

Furthermore, we need to specify an underlying metric for measuring distances. We decided to use the L^∞-metric, which defines the distance of two vertices u and v to be $\max(|x(u) - x(v)|, |y(u) - y(v)|)$. This metric has the nice property that all points on the boundary of the unit square centered at any point p have the same distance from p. In Figure 6.1 eight points on the octilinear coordinate axes are shown that all have the same L^∞-distance from the origin. One side-effect of this is that all vertices will be placed on a rectilinear grid as long as all edge lengths in the L^∞-metric are integers. Attention needs to be paid because a z_1- or z_2-coordinate difference of 2 corresponds to a L^∞-distance of 1.

6.1.2 Octilinear Edges

First of all, this part of the constraints models that all edges are drawn as straight, octilinear line segments that conform to their respective minimum lengths, as stated in hard constraints (H2) and (H3). Secondly, we also restrict the possible octilinear directions for each edge. In octilinear drawings without further restrictions, each edge uv can be drawn in eight different directions when fixing one of its endpoints. Although geographic exactness is less important in metro maps, it should not be ignored completely. This is formulated in soft constraint (S3), which means for example that if a metro line is running in a northern direction in reality then we do not want to draw it running southwards. Although it would be possible to model the *relative position* solely as part of the objective function, we suggest to restrict the edge directions to a set of acceptable directions and only consider those directions in the objective function.

Before formulating the constraints we need some notation to address relative positions between vertices and to denote directions of edges. As depicted in Figure 6.2, we define for each vertex v a partition of the plane into eight sectors. Each sector is a 45-degree wedge with apex v. The wedges are centered around rays that emanate from v and follow one of the four orientations either in positive or in negative direction. The sectors are numbered from 0 to 7 counterclockwise starting with the positive x-direction.

To denote the rough relative position between two vertices u and v in the *original* layout we use the terms $\text{sec}_u(v)$ and $\text{sec}_v(u)$ representing the sector relative to u in which v lies and vice versa. Similarly, for each edge uv, we define a variable $\text{dir}(u, v)$ to denote the octilinear direction of uv relative to u in the *new* layout. We identify each octilinear direction with its corresponding sector. For example if edge uv leaves u in negative z_1-direction, we say $\text{dir}(u, v) = 5$. Note that the following equalities hold: $\text{sec}_u(v) = \text{sec}_v(u) + 4 \pmod 8$ and $\text{dir}(u, v) = \text{dir}(v, u) + 4 \pmod 8$.

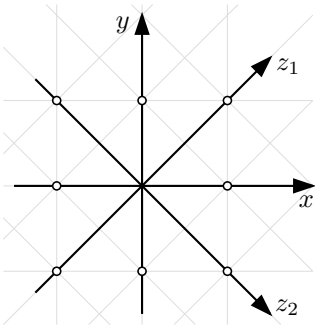


Figure 6.1: The octilinear coordinate system with an indicated octilinear grid in the background. The marked points all have the same L^∞-distance from the origin.

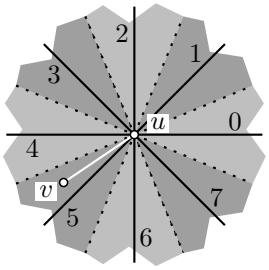


Figure 6.2: Numbering of the sectors and the octilinear directions relative to vertex u, e.g. $\text{sec}_u(v) = 5$.

As mentioned in the beginning of this section, we partially model the soft constraint (S3) as a hard constraint. As a compromise between strong conservation of relative positions and flexibility to obtain a "nice" drawing, we allow that an edge is drawn in three different ways. It can be drawn in the direction corresponding to its original sector relative to either endpoint or it can be drawn in the two neighboring directions (and thus the maximum deviation from the original direction is 67.5°). Let $\text{sec}_u^{\text{pred}}(v) = \text{sec}_u(v) - 1 \pmod 8$, $\text{sec}_u^{\text{orig}}(v) = \text{sec}_u(v)$ and $\text{sec}_u^{\text{succ}}(v) = \text{sec}_u(v) + 1 \pmod 8$. Recall that the values $\text{sec}_u^i(v)$ are fixed and only depend on the original geographic position of the vertices, so there is no need to perform modulo operations inside the MIP. Now, we restrict the variables of type $\text{dir}(u, v)$, which will be used in Sections 6.1.3, 6.2.2 and 6.2.3, to the set $\{\text{sec}_u^{\text{pred}}(v), \text{sec}_u^{\text{orig}}(v), \text{sec}_u^{\text{succ}}(v)\}$. This is expressed by the disjunction

$$\bigvee_{i \in \{\text{pred,orig,succ}\}} (\text{dir}(u, v) = \text{sec}_u^i(v) \ \wedge \ \text{dir}(v, u) = \text{sec}_v^i(u)) \tag{6.2}$$

To model the choice of direction we introduce three binary variables $\alpha_{\text{pred}}, \alpha_{\text{orig}}, \alpha_{\text{succ}}$ and the constraint

$$\alpha_{\text{pred}}(u, v) + \alpha_{\text{orig}}(u, v) + \alpha_{\text{succ}}(u, v) \quad = \quad 1 \qquad \forall uv \in E. \tag{6.3}$$

There is exactly one variable in this constraint that takes the value 1 and this variable will determine the direction in which edge uv is drawn, i.e. the term of disjunction (6.2) that will evaluate to true.

Now, for each edge uv, we model the correct assignment of $\text{dir}(u, v)$ and $\text{dir}(v, u)$. For each $i \in \{\text{pred}, \text{orig}, \text{succ}\}$ we have the following set of constraints

$$\begin{array}{rcl} \text{dir}(u, v) - \text{sec}_u^i(v) & \leq & M(1 - \alpha_i(u, v)) \\ -\text{dir}(u, v) + \text{sec}_u^i(v) & \leq & M(1 - \alpha_i(u, v)) \end{array} \qquad \forall uv \in E \tag{6.4}$$

for $\text{dir}(u, v)$ and analogously the constraints

$$\begin{array}{rcl} \text{dir}(v, u) - \text{sec}_v^i(u) & \leq & M(1 - \alpha_i(u, v)) \\ -\text{dir}(v, u) + \text{sec}_v^i(u) & \leq & M(1 - \alpha_i(u, v)) \end{array} \qquad \forall uv \in E \tag{6.5}$$

for $\text{dir}(v, u)$, where these variables of type $\text{dir}(u, v)$ are integers in the range $\{0, \ldots, 7\}$ and M is a large constant. Note that only one binary variable of type $\alpha_i(u, v)$ is used for both constraints (6.4) and (6.5).

$\mathrm{dir}(u,v)$	condition 1			condition 2		
0	$y(u)$	$=$	$y(v)$	$x(v)-x(u)$	$\geq$	ℓ_{uv}
1	$z_2(u)$	$=$	$z_2(v)$	$z_1(v)-z_1(u)$	$\geq$	$2\ell_{uv}$
2	$x(u)$	$=$	$x(v)$	$y(v)-y(u)$	$\geq$	ℓ_{uv}
3	$z_1(u)$	$=$	$z_1(v)$	$z_2(u)-z_2(v)$	$\geq$	$2\ell_{uv}$
4	$y(u)$	$=$	$y(v)$	$x(u)-x(v)$	$\geq$	ℓ_{uv}
5	$z_2(u)$	$=$	$z_2(v)$	$z_1(u)-z_1(v)$	$\geq$	$2\ell_{uv}$
6	$x(u)$	$=$	$x(v)$	$y(u)-y(v)$	$\geq$	ℓ_{uv}
7	$z_1(u)$	$=$	$z_1(v)$	$z_2(v)-z_2(u)$	$\geq$	$2\ell_{uv}$

Table 6.1: The octilinearity conditions for edge uv depending on the specified direction $\mathrm{dir}(u,v)$. In the MIP these are formulated similarly to constraints (6.6).

The use of the large constant M in connection with a set of binary variables as in constraint (6.3) is a standard trick in MIP modeling for formulating a disjunction of constraints. The constant M must be an upper bound on the left-hand sides of the inequalities. For example in the set of constraints (6.4) and (6.5) it is sufficient to set M equal to 8. Choosing M as small as possible can reduce the runtime of the MIP solver. However, for ease of presentation we will simply use M to denote some sufficiently large constant. Here, if $\alpha_i(u,v)=0$, the four constraints in (6.4) and (6.5) are trivially fulfilled and have no influence on the values of the variables that appear on the left-hand sides. On the other hand if $\alpha_i(u,v)=1$, the four inequalities are equivalent to $\mathrm{dir}(u,v)=\mathrm{sec}_u^i(v)$ and $\mathrm{dir}(v,u)=\mathrm{sec}_v^i(u)$ as desired (equality constraints have to be transformed into two inequalities when using the trick with the large constant). Due to constraint (6.3), it holds that $\alpha_i(u,v)=1$ for exactly one $i\in\{\mathrm{pred},\mathrm{orig},\mathrm{succ}\}$. Thus, exactly one term of the disjunction (6.2) is fulfilled.

Further, depending on the actual values of $\mathrm{sec}_u^i(v)$, we formulate three more constraints for each $i\in\{\mathrm{pred},\mathrm{orig},\mathrm{succ}\}$. For example let $\mathrm{sec}_u^{\mathrm{orig}}(v)=2$ (meaning v is vertically above u in the original layout). Then the constraints for edge uv and $i=\mathrm{orig}$ are as follows

$$\begin{aligned} x(u)-x(v) &\leq M(1-\alpha_{\mathrm{orig}}(u,v)) \\ -x(u)+x(v) &\leq M(1-\alpha_{\mathrm{orig}}(u,v)) \\ -y(u)+y(v) &\geq -M(1-\alpha_{\mathrm{orig}}(u,v))+\ell_{uv}, \end{aligned} \tag{6.6}$$

where $\ell_{uv}>0$ is the minimum length of edge uv. If $\alpha_{\mathrm{orig}}(u,v)=1$, these constraints force u and v to have the same x-coordinate and to keep a vertical distance of at least ℓ_{uv}. This is exactly what is needed for an edge running vertically upwards. The other seven possibilities are formulated similarly by forcing one of the coordinates of both vertices to be equal and the distance along the respective octilinear direction to be at least ℓ_{uv} as given in Table 6.1. Recall from Section 6.1.1 that for our metric the lengths in diagonal directions need to be multiplied by 2. For $i=\mathrm{pred}$ or $i=\mathrm{succ}$ the constraints are constructed analogously.

This part of our MIP formulation consists of $22m$ constraints and $5m$ variables in total.

6.1.3 Preservation of the Embedding

To guarantee preservation of the original embedding as demanded in hard constraint (H1) it suffices in a planar drawing to maintain for each vertex $v \in V$ the circular ordering of all incident edges.

Let $N(v) = \{u_1, u_2, \ldots, u_{\deg(v)}\}$ denote the set of all neighbors of v. The counterclockwise ordering of the edges $vu \in E$ incident to v implies an ordering on $N(v)$ by identifying each edge vu with the vertex u opposite of v. Assume that the ordering is $u_1 < u_2 < \ldots < u_{\deg(v)}$. We have to ensure that this ordering is maintained. In the metro-map layout one of these vertices, say u_j (more precisely the variable $\mathrm{dir}(v, u_j)$), is assigned the smallest direction number from the set of possible directions $\{0, \ldots, 7\}$ using the coding scheme introduced in the previous section. This means that u_j is encountered first when rotating a ray counterclockwise around v starting in positive x-direction. All other vertices in $N(v)$ must follow in the order specified by the input embedding and they must have strictly increasing direction numbers:

$$\mathrm{dir}(v, u_j) < \mathrm{dir}(v, u_{j+1}) < \ldots < \mathrm{dir}(v, u_{j+\deg(v)-1}),$$

where all indices greater than $\deg(v)$ are considered modulo $\deg(v)$ for the rest of this section. In other words, all but one of the inequalities

$$\begin{array}{rcl} \mathrm{dir}(v, u_1) & < & \mathrm{dir}(v, u_2), \\ \mathrm{dir}(v, u_2) & < & \mathrm{dir}(v, u_3), \\ & \vdots & \\ \mathrm{dir}(v, u_{\deg(v)-1}) & < & \mathrm{dir}(v, u_{\deg(v)}), \\ \mathrm{dir}(v, u_{\deg(v)}) & < & \mathrm{dir}(v, u_1) \end{array}$$

must hold.

In order to model the selection of the vertex with smallest direction number, we again use binary variables as in Section 6.1.2. But instead of applying the standard trick to model a disjunction of $\deg(v)$ many terms with $\deg(v) - 1$ constraints each, we make use of the fact that in each case exactly one of the inequalities may be violated while the rest must hold. This requires about a factor $\deg(v)$ less constraints. They are as follows:

$$\beta_1(v) + \beta_2(v) + \ldots + \beta_{\deg(v)}(v) = 1 \qquad \forall v \in V,\ \deg(v) \geq 2, \tag{6.7}$$

with binary variables $\beta_i(v)$, and

$$\begin{array}{rcl} \mathrm{dir}(v, u_2) - \mathrm{dir}(v, u_1) & \geq & -M\beta_1(v) + 1 \\ \mathrm{dir}(v, u_3) - \mathrm{dir}(v, u_2) & \geq & -M\beta_2(v) + 1 \\ & \vdots & \\ \mathrm{dir}(v, u_1) - \mathrm{dir}(v, u_{\deg(v)}) & \geq & -M\beta_{\deg(v)}(v) + 1 \end{array} \qquad \forall v \in V,\ \deg(v) \geq 2. \tag{6.8}$$

For each $\beta_i(v)$ with $i = 1, \ldots, \deg(v)$ there is a corresponding constraint in the list (6.8) where $\beta_i(v)$ appears on the right-hand side. All but one $\beta_i(v)$ are set to 0 by constraint (6.7). The inequalities corresponding to those $\beta_i(v)$ that are equal to 0 hence read as

$$\mathrm{dir}(v, u_{i+1}) - \mathrm{dir}(v, u_i) \geq 1.$$

This simply means that the direction values must satisfy $\mathrm{dir}(v, u_i) < \mathrm{dir}(v, u_{i+1})$ as required by the given circular ordering. For one index j the value of $\beta_j(v)$ equals 1. Now the corresponding inequality reads as follows

$$\mathrm{dir}(v, u_{j+1}) - \mathrm{dir}(v, u_j) \geq -M + 1$$

and is trivially satisfied for any left-hand side. Hence, vertex u_{j+1} is determined as the vertex with smallest direction number among $N(v)$ because its direction number does not have to be larger than the one of its predecessor. Altogether, the above constraints assure that the input embedding is preserved.

The constraints in this part not only enforce that the combinatorial embedding is preserved but also that no two edges incident to the same vertex can have the same direction. An upper bound on the number of constraints and variables for this part of the MIP is given by $\sum_{v \in V}(\deg(v) + 1)$ which is in $O(m)$.

6.1.4 Planarity

To guarantee planarity we have to ensure that pairs of edges do not intersect. This can be done for a pair (e_1, e_2), $e_1 \neq e_2$ of straight-line octilinear edges by distinguishing eight possible relative positions for them. We express these relative positions using compass orientations. Fixing one edge $e_1 = u_1v_1$ the second edge $e_2 = u_2v_2$ must be placed *north*, *south*, *east*, *west* or *northeast*, *northwest*, *southeast*, *southwest* of e_1. For example northeast means in terms of our coordinate system that both vertices incident to e_1 have strictly smaller z_1-values than both vertices incident to e_2. The other terms are defined in a similar way. By setting the minimum distance in the respective direction to $d_{\min}$ we also model hard constraint (H4).

Clearly, an octilinear straight-line drawing is planar if and only if all pairs of non-incident edges satisfy at least one of the above conditions. Naively, we model this disjunctive constraint indeed for all pairs of non-incident edges as follows:

$$\sum_{i \in \{\text{N,S,E,W,NE,NW,SE,SW}\}} \gamma_i(e_1, e_2) \geq 1 \qquad \begin{array}{l} \forall (e_1, e_2) \in E \times E, \\ e_1, e_2 \text{ not incident,} \end{array} \tag{6.9}$$

where the variables $\gamma_{\text{N}}, \ldots, \gamma_{\text{SW}}$ are binary. With the help of these binary variables and the constant M the selection of the corresponding relative position is modeled. As an example we give the constraints for the condition "e_2 is east of e_1"

$$\begin{array}{rcll} x(u_1) - x(u_2) & \leq & M(1 - \gamma_E(e_1, e_2)) - d_{\min} & \\ x(u_1) - x(v_2) & \leq & M(1 - \gamma_E(e_1, e_2)) - d_{\min} & \forall (e_1, e_2) \in E \times E, \\ x(v_1) - x(u_2) & \leq & M(1 - \gamma_E(e_1, e_2)) - d_{\min} & e_1, e_2 \text{ not incident.} \\ x(v_1) - x(v_2) & \leq & M(1 - \gamma_E(e_1, e_2)) - d_{\min} & \end{array} \tag{6.10}$$

Recall that $d_{\min}$ is the minimum distance between non-incident edges as given in hard constraint (H4). Analogously, each of the other seven relative positions is modeled using four constraints each. This amounts to a total of 33 constraints and 8 binary variables for each edge pair. The problem is that the number of possible non-incident edge pairs is $O(m^2)$ and hence these constraints and variables are responsible for more than 90 percent of the MIP size in practice (see Chapter 7 for the actual numbers). Therefore, in Section 6.3 we give several heuristics to reduce the number of constraints that enforce planarity.

6.2 Objective Function

The goal of the objective function is to model the soft constraints (S1)–(S3) such that minimizing the objective function corresponds to optimally satisfying the soft

constraints. Since we are considering three different soft constraints we need to create individual linear objective functions for each of them. These will then be optimized simultaneously using a weighted sum of the three objective functions:

$$\text{Minimize } \lambda_{\text{length}} \, \text{cost}_{\text{length}} + \lambda_{\text{bends}} \, \text{cost}_{\text{bends}} + \lambda_{\text{relpos}} \, \text{cost}_{\text{relpos}}, \tag{6.11}$$

where the variables of type λ_i are positive user-defined weights, each of which individually emphasizes a certain esthetics criterion. Like all multicriteria optimization techniques the individual criteria might be conflicting terms and thus the result will usually be a compromise of the individual goals. Consequently, adjusting the weights is an important task that needs to be done carefully. Next, we describe the three cost functions and additional constraints needed by the respective cost functions. Finally, in Section 6.2.4, we summarize our MIP model for drawing metro maps.

6.2.1 Minimizing Edge Lengths

The edge lengths are given in the L^∞-metric as stated before. So, for modeling the soft constraint (S2) we define new real-valued, non-negative variables $D(u, v)$ for all edges $uv \in E$ which serve as upper bounds on the lengths of their respective edges. By setting

$$\text{cost}_{\text{length}} = \sum_{uv \in E} D(u, v) \tag{6.12}$$

and by minimizing $\text{cost}_{\text{length}}$, these variables will become tight upper bounds and indeed equal the corresponding edge lengths.

The constraints that bound $D(u, v)$ depend on the actual direction of edge uv. Note that this direction is determined according to the constraints in Section 6.1.2. Thus we can reuse the binary variables defined in constraint (6.3) in that section to distinguish the three cases for the edge direction. The binary variables tell us which direction is selected and we can use the appropriate coordinate differences to determine the edge length. As an example assume that $\text{sec}_u(v) = 1$. Then the constraints for edge uv are

$$\begin{aligned} x(v) - x(u) &\leq M(1 - \alpha_{\text{pred}}(u, v)) + D(u, v) \\ x(v) - x(u) &\leq M(1 - \alpha_{\text{orig}}(u, v)) + D(u, v) \\ y(v) - y(u) &\leq M(1 - \alpha_{\text{succ}}(u, v)) + D(u, v) \end{aligned} \tag{6.13}$$

Note that for a diagonal edge uv it holds that $|x(u) - x(v)| = |y(u) - y(v)|$. Hence, in this case we can use either the x- or the y-coordinates to determine the length $D(u, v)$. This also avoids the multiplication by a factor 2 which would be otherwise necessary for diagonal lengths. The edge lengths for other values of $\text{sec}_u(v)$ are modeled similarly by using the right coordinate differences on the left-hand sides of Equation (6.13). In total we need m variables and $3m$ constraints.

6.2.2 Avoiding Line Bends

Clarity in an octilinear drawing depends crucially on the ability to visually follow the metro lines. This can be partially enhanced by using distinguishable colors, but also by avoiding bends along the lines as formulated in soft constraint (S1).

We define the bend cost subject to the actual angle between two adjacent edges on a path $L \in \mathcal{L}$. Due to the octilinearity constraints and to the fact that two adjacent edges cannot have the same direction relative to their joint vertex the

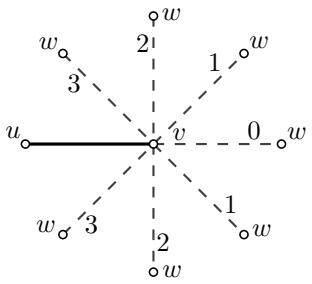


Figure 6.3: Bend cost $\text{bend}(u, v, w)$ for each value of $\text{dir}_v(w)$. The cost increases with the acuteness of the angle between edges uv and vw.

angles can only equal 180, 135, 90, and 45 degrees. In that order we define the corresponding bend cost to be 0, 1, 2, and 3, such that the cost increases with the acuteness of the angle, see Figure 6.3.

In our model we can determine the angle between two adjacent edges uv and vw by using the values of $\text{dir}(u, v)$ and $\text{dir}(v, w)$ that have been assigned by the constraints in Section 6.1.2. For ease of notation let $\Delta\text{dir}(u, v, w) = \text{dir}(u, v) - \text{dir}(v, w)$. Note that the value of $\Delta\text{dir}(u, v, w)$ ranges from -7 to 7. Then, the bend cost can be expressed as

$$\text{bend}(u, v, w) = \begin{cases} |\Delta\text{dir}(u, v, w)| & \text{if } |\Delta\text{dir}(u, v, w)| \leq 4 \\ 8 - |\Delta\text{dir}(u, v, w)| & \text{if } |\Delta\text{dir}(u, v, w)| \geq 5. \end{cases} \tag{6.14}$$

Now we can define the total bend cost of the drawing as

$$\text{cost}_{\text{bends}} = \sum_{uv, vw \in L,\ L \in \mathcal{L}} \text{bend}(u, v, w). \tag{6.15}$$

Minimizing this value hence minimizes the number and acuteness of all bends along all lines in $\mathcal{L}$. An intensified version could assign higher, e.g. double, costs to bends that appear in intersection vertices to stress that lines should go straight through those vertices.

The formulation of bend cost in (6.14) cannot be transformed directly into a set of linear constraints because it involves absolute values and a case distinction. Here, we solve this problem using instead the following constraints for all lines $L \in \mathcal{L}$ and pairs of incident edges uv, vw on L. Again, we need some binary variables, namely $\delta_1(u, v, w), \delta_2(u, v, w)$, and $\delta_3(u, v, w)$. The constraint

$$\delta_1(u, v, w) + \delta_2(u, v, w) + \delta_3(u, v, w) \quad = \quad 2 \tag{6.16}$$

makes sure that exactly one of the three binary variables takes the value 0. Then, the set of constraints

$$\begin{array}{rcl} \Delta\text{dir}(u, v, w) & \leq & -5 + \delta_1(u, v, w)M \\ \Delta\text{dir}(u, v, w) & \geq & 5 - \delta_2(u, v, w)M \\ \Delta\text{dir}(u, v, w) & \leq & 4 + \delta_3(u, v, w)M \\ \Delta\text{dir}(u, v, w) & \geq & -4 - \delta_3(u, v, w)M \end{array} \tag{6.17}$$

establishes the following relationship between the value of $\Delta\text{dir}(u, v, w)$ and the variables of type $\delta_i(u, v, w)$:

$$\begin{array}{rcl} & \Delta\text{dir}(u, v, w) \leq -5 \Leftrightarrow \delta_1(u, v, w) = 0, & \\ & \Delta\text{dir}(u, v, w) \geq 5 \Leftrightarrow \delta_2(u, v, w) = 0, & \\ -4 \leq & \Delta\text{dir}(u, v, w) \leq 4 \Leftrightarrow \delta_3(u, v, w) = 0. & \end{array} \tag{6.18}$$

The bend cost $\text{bend}(u, v, w)$ is meant to take values in the range from 0 to 4 for all possible values of $\Delta\text{dir}(u, v, w)$ as defined in (6.14). For $\delta_3(u, v, w) = 0$ the bend cost equals $|\Delta\text{dir}(u, v, w)|$. However, for $\delta_1(u, v, w) = 0$ or $\delta_2(u, v, w) = 0$ this equality does not hold. Note that the values 5, 6, and 7 for $|\Delta\text{dir}(u, v, w)|$ in these cases correspond to bend costs of 3, 2, and 1 in that order. So subtracting or adding 8 to $\Delta\text{dir}(u, v, w)$, depending on the sign, before determining its absolute value establishes the desired bend cost. This is modeled by the two constraints

$$\begin{array}{rcl} \Delta\text{dir}(u, v, w) - 8\delta_1(u, v, w) + 8\delta_2(u, v, w) & \geq & -\text{bend}(u, v, w) \\ \Delta\text{dir}(u, v, w) - 8\delta_1(u, v, w) + 8\delta_2(u, v, w) & \leq & \text{bend}(u, v, w) \end{array} \tag{6.19}$$

that assign the bend cost $\text{bend}(u, v, w)$ for the bend between edges uv and vw. The variable $\text{bend}(u, v, w)$ is integer valued and non-negative. These two constraints make $\text{bend}(u, v, w)$ an upper bound on the absolute value of the left-hand sides in (6.19). By minimizing $\text{bend}(u, v, w)$ inside $\text{cost}_{\text{bends}}$, the bound gets tight and thus equal to that absolute value. For $\delta_3(u, v, w) = 0$ (and thus $\delta_1(u, v, w) = 1$ and $\delta_2(u, v, w) = 1$) we add and subtract 8 at the same time on the left-hand sides. However, if one of $\delta_1(u, v, w)$ or $\delta_2(u, v, w)$ equals 0 then the addition or subtraction of 8 really takes place and hence establishes the correct value of the left-hand side as described before. All in all, this models the bend cost as defined in (6.14).

Minimizing the number of bends thus uses four variables and seven constraints for each pair of incident edges along a path $L \in \mathcal{L}$. Since there are in total at most m' such pairs we are using at most $4m'$ variables and $7m'$ constraints.

6.2.3 Preserving Relative Positions

To preserve as much of the overall appearance of the metro system as possible we have restricted the edge directions to the set of the three directions closest to the original one in Section 6.1.2. This already avoids that the new layout drastically alters relative positions between adjacent vertices. Ideally, we want to draw an edge uv using its nearest octilinear approximation, i.e. the direction where $\text{dir}(u, v) = \text{sec}_u(v)$. We introduce a cost of 1 in case that the layout does not use that direction. If we had not restricted each edge to a set of three directions at an earlier stage a more sophisticated cost function would be necessary that charges the amount of deviation from the optimal direction. In our simple case, a binary cost function suffices. This models soft constraint (S3) and prefers layouts that preserve the relative positions.

For each edge uv we define as its relative-position cost a binary variable $\epsilon(u, v)$ which is 0 if and only if $\text{dir}(u, v) = \text{sec}_u(v)$. This is modeled as follows

$$-M\epsilon(u, v) \leq \text{dir}(u, v) - \text{sec}_u(v) \leq M\epsilon(u, v) \qquad \forall uv \in E. \tag{6.20}$$

Now we can define the cost for deviating from the original relative positions as

$$\text{cost}_{\text{relpos}} = \sum_{uv \in E} \epsilon(u, v) \tag{6.21}$$

which, for each edge, charges 1 when not selecting the nearest octilinear direction. We need m variables and $2m$ constraints for this part of the objective function.

6.2.4 Summary of the Model

We now summarize the constraints given in Sections 6.1.1–6.1.4 and the objective functions given in Sections 6.2.1–6.2.3. Taken together, they constitute a MIP that

solves the NP-hard metro-map layout problem. If the input graph is metro-map drawable our MIP finds a layout that satisfies all hard constraints (H1)–(H4) and at the same time optimizes a weighted sum of cost functions corresponding to the soft constraints (S1)–(S3). Otherwise the MIP is infeasible. The following theorem summarizes the number of constraints and variables.

Theorem 6.1 *The Metro-Map Layout Problem can be formulated as mixed-integer program using $O(n + m' + m^2)$ variables and linear constraints.*

Note that only the planarity constraints in Section 6.1.4 require $O(m^2)$ variables and constraints. Otherwise the model is of linear size. This motivates the development of planarity heuristics in the next section.

6.3 Speed-Up Heuristics

Driven by the large size of the mixed-integer programs for our real-world examples and by the extremely low speed of the optimizer on these large programs, we decided to implement some heuristics to reduce the MIP size and hence speed up the computation of our layouts. The first idea reduces the size of the metro graph by removing some dispensable vertices and the second heuristic cuts down the number of constraints in charge of guaranteeing planarity. Recall that this number is quadratic in the input size.

6.3.1 Reducing the Graph Size

A common feature of real-world metro maps is that they tend to have a large number of degree-2 vertices on routes between two interchange stations. It is useful and common in real metro maps to treat a path between pairs of neighboring interchange or terminal stations as a whole and draw it as straight as possible. It adds unnecessary complexity to the model when such a path needs to be straightened edge by edge. This leads to the idea of removing chains of degree-2 vertices and replacing them by a single edge. The algorithm must be aware of the fact that such an edge represents multiple short edges. The simplest way to do so is to set the minimum length of the new edge equal to the number of vertices that had been removed during its creation. In the final drawing the removed vertices can be reinserted equidistantly on their respective edges.

This data-reduction trick has been applied before in the context of metro maps [HMN05, SR04] but it lacks some flexibility because it requires that all interchange stations must be connected by *straight*, octilinear line segments. This might either distort the layout in order to be drawable or there might not even exist a solution.

To this end we extend the simple approach and retain up to two vertices on each chain of degree-2 vertices. These vertices act as joints such that the path between two interchanges can be drawn as a polyline with three segments or *links*. Hence, we call this method *3-link heuristic*. This allows that interchanges can get connected with much less distortion of the remaining graph. Again, the removed degree-2 vertices are reinserted equidistantly on the respective polylines in the final drawing. Our experiments show that using 3-link polylines is a good compromise between flexibility of the drawing and size of the MIP model. Since the target function penalizes bends along lines the possibility to introduce bends is in fact rarely used. Note that Cabello et al. [CBD+01] also apply 3-link polylines to connect vertices in schematic maps.

There are some technical details in connection with this preprocessing step. Degree-2 vertices can only be collapsed as long as no multi-edges are created. The definition of collapsing such a vertex (see Section 2.1.1) considers this case. When two edges are merged by the removal of their common vertex their lengths are added and the sum is set as the new length to accommodate all degree-2 vertices. After contracting chains of degree-2 vertices in this way all the meandering of this part of the line has disappeared since the direction of an edge is computed by the positions of its end vertices only.

In the implementation of the 3-link reduction method we do not collapse vertices that are directly adjacent to interchange vertices. This means that on subpaths of a line L between two interchanges, which consist of three or more degree-2 vertices, the two outermost degree-2 vertices are retained. Shorter paths are not modified. For subpaths that lead to termini, i.e. degree-1 vertices, only the degree-2 vertex next to the interchange is retained. However, there is no reason to place the bends on such a 3-link (and similarly for a 2-link) path close to the ends, which would be the case here since only the edge in the middle of the path represents collapsed vertices. Therefore, we set the minimum length of each individual edge to 1, but also place a lower bound proportional to the number of collapsed vertices on the sum of all three edge lengths. So, there is no longer a bias as to where the bends can be placed along the subpath while it is still guaranteed that all degree-2 vertices can be reinserted.

6.3.2 Planarity Heuristics

Our MIP formulation consists of several parts among which the only part that needs a quadratic number of constraints and variables is the part that ensures planarity. This is why we suggest several ways to reduce the size of this part.

For a planar drawing of an embedded graph it suffices to require that non-incident edges belonging to the same face do not intersect. This already guarantees that no two edges intersect except at common endpoints. So instead of using the constraints in Section 6.1.4 for *all* pairs of non-incident edges we only include them in the MIP for pairs of non-incident edges of the same face. This *face method* should always be applied as it reduces the MIP size while still guaranteeing that all the hard constraints are satisfied as before. Yet, the worst-case size of the planarity constraints and variables is still $O(m^2)$.

In many real world examples (see Chapter 7) this reduction is not enough to solve the MIP in an acceptable amount of time. To further reduce the number of constraints we have to rely on heuristic methods that relax the planarity requirements and may lead to non-planar drawings in some cases.

One extreme concerning the size of the planarity part is to skip the planarity constraints completely. The hope behind this is that the structure of the metro graph is such that minimizing the number of bends and optimizing the relative position suffices to create a planar drawing. This means that intersections only appear when lines deviate strongly from their geographic location or have more bends than necessary. Indeed, our experiments in the next chapter show that for many real-world examples the result is a planar drawing in spite of not being enforced in the MIP.

Still, sometimes we need to avoid intersections explicitly. We implemented two ideas to identify critical edge pairs, i.e. edge pairs that are likely to intersect. The first observation is that on the one hand the external face of the metro graph has usually far more vertices than any other face. On the other hand it seems

unlikely in a drawing minimizing geographic distortion that for example edges on the left side of the external face in the original layout intersect those edges originally on the right side. Inspired by this observation we construct the convex hull of the input layout and add dummy edges between adjacent polygon vertices on the convex hull. This partitions the set of edges of the external face into several subsets which we can use instead of the external face itself when checking pairs of edges for intersection. These dummy edges only serve to partition the large external face and are removed immediately after generating the planarity constraints. This method is called *convex-hull heuristic*.

The second observation from our experiments is that most of the time intersections involve pendant edges, i.e. edges that lead to degree-1 vertices. One reason might be that a pendant edge is not tightened at two points but only at one point, such that the loose end can move across other edges. Hence, a second heuristic step is to consider only pairs of edges where at least one edge is a pendant edge. This *pendant-edge heuristic* can be combined optionally with the convex-hull heuristic. The combination of both heuristics sufficed in all our experiments to generate planar layouts and managed to drastically reduce the MIP size.

6.4 Label Placement

Metro maps in practice are of little interest to a passenger of the metro system unless all stations are labeled by their name. Because labels require space in the layout they should be considered in a drawing algorithm right from the beginning. In this section we show how to model and implement station labels in a MIP for the special case that we collapse all degree-2 vertices as described in Section 6.3.1.

The general idea is to model all labels for collapsed degree-2 or degree-1 vertices along an edge as a whole. The vertex labels will be placed inside a parallelogram-shaped region that is attached to the corresponding edge. The side length of the parallelogram matches the length of the longest vertex label. This trick also makes sure that all labels of stations on one edge are consistently placed on the same side of that edge. Both to keep the number of reading directions small and to avoid unnecessary complexity in the model we restrict labels to be placed horizontally or, if the corresponding edge itself is horizontal, diagonally in z_1-direction. We modify the given metro graph by adding new vertices and edges such that each parallelogram forms a new special face. As our MIP creates a planar drawing of this extended metro graph, we can also place the labels without overlap. Binucci et al. [BDLN05] use a similar model to label edges in an orthogonal drawing with rectangles. However, they consider simple edge labels and do not use these rectangles for labeling a set of collapsed vertices. Labels of interchanges are modeled individually as an edge of length equal to the label length. This labeling approach cannot be simply extended to the 3-link case because, then, we do not know in advance which vertex will be positioned on which of the three links. This, however, would be necessary to model groups of labels as parallelograms.

In the following we give the details of our model for edge labels. Let us pick a non-horizontal edge $e = uv$ as depicted in Figure 6.4. To label all degree-2 vertices on e we first insert two dummy vertices u_1, u_2 on e next to the endpoints of e. Then, we make sure that e is not allowed to bend at u_1 and u_2. This is done by adding the constraints

$$\begin{array}{rcl} \mathrm{dir}(u, u_1) & = & \mathrm{dir}(u_1, u_2) \\ \mathrm{dir}(u_2, v) & = & \mathrm{dir}(u_1, u_2) \end{array} \tag{6.22}$$

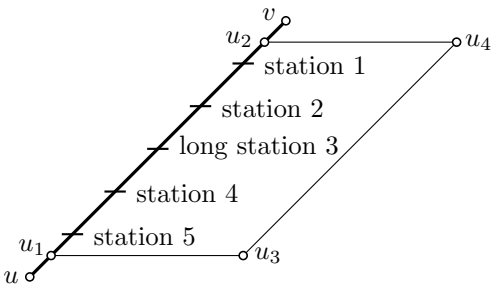


Figure 6.4: Modeling vertex labels with a parallelogram-shaped region attached to edge uv. The area fits all labels of the four vertices indicated with tickmarks.

to the MIP. We add two more vertices u_3, u_4 and the edges u_1u_3, u_3u_4, u_2u_4. Edges u_1u_3 and u_2u_4 are constrained to be horizontal and of length at least $\ell_{u_1u_3}$. For u_1u_3 this is done with the following constraints

$$y(u_1) \quad = \quad y(u_3) \tag{6.23}$$

and

$$\begin{array}{rcl} x(u_1) - x(u_3) & \leq & (1-\zeta(uv))M + \ell_{u_1u_3} \\ x(u_1) - x(u_3) & \geq & -(1-\zeta(uv))M + \ell_{u_1u_3} \\ x(u_3) - x(u_1) & \leq & \zeta(uv)M + \ell_{u_1u_3} \\ x(u_3) - x(u_1) & \geq & -\zeta(uv)M + \ell_{u_1u_3}, \end{array} \tag{6.24}$$

where $\zeta(uv)$ is a binary variable that selects whether the labels are on the left or right side of e. The length $\ell_{u_1u_3}$ equals the length of the longest vertex label on edge e. For edge u_2u_4 the constraints are given analogously to (6.23) and (6.24) using the same binary variable $\zeta(uv)$. (Similarly, edges u_1u_3 and u_2u_4 are constrained to be z_1-diagonal if e is a horizontal edge.) The third edge u_3u_4 is constrained to be parallel to u_1u_2 by the constraint

$$\mathrm{dir}(u_3, u_4) = \mathrm{dir}(u_1, u_2) \tag{6.25}$$

so that the four new edges indeed form a parallelogram attached to e. This parallelogram can still be placed on either side of e, which is modeled with the binary variable $\zeta(uv)$. Clearly, we must skip the constraints to preserve an embedding around vertices u_1 and u_2 because they are meant to have a variable order of their incident edges. Also, the new edges u_1u_3, u_3u_4, and $u2_u4$ do not belong to any metro lines, and neither their bends nor their lengths are considered in the objective function. Finally, because edges can have three possible directions, we need to distinguish this for the labels too since labels on horizontal edges are treated differently from labels on non-horizontal edges. We deal with this case distinction using another binary variable that selects between the sets of constraints for horizontal and for diagonal labels accordingly.

It remains to show how to label intersection vertices. Let v be a vertex with $3 \leq \deg(v) \leq 7$. We model the label of v simply as an edge vw, where w is a new vertex. The length ℓ_{vw} is set equal to the length of the label of v. For consistency with the other labels the direction of edge vw is restricted to the set $\{0, 1, 4, 5\}$, which corresponds to a horizontal or z_1-diagonal edge. The constraints that preserve the circular order of $N(v)$ are modified such that vertex w can appear anywhere in that order. However, $\mathrm{dir}(v, w)$ cannot be equal to $\mathrm{dir}(v, u)$ for any $u \in N(v)$. Obviously, we cannot label vertices of degree 8 yet because there is no space for an additional edge left. However, no such vertex occured in any of our real-world examples.

We have implemented a first version of the MIP including label placement. There are still some tricky technicalities to deal with in order to get the planarity heuristics to work correctly for these modified metro graphs. This is mainly caused by the fact that the additional label edges can belong to different faces, depending on the actual label direction. Of course, the introduction of new dummy vertices and edges to model labels increases the size of the metro graph and consequently the size of the MIP. Section 7.6 shows an example of a labeled metro map.

6.5 Implementation

We implemented our algorithm in the Java programming language. The program is a command-line tool that requires some user interaction. Creating a metro map layout involves three steps as indicated in Figure 6.5. In the beginning the metro graph is read in from a file. Then, after selecting how to preprocess the graph and which heuristics to apply, the MIP is generated and written to a file in a standard format for linear programs. This file can then be read and solved by any external MIP optimizer. The optimizer's solution file is opened by the Java program again and the corresponding layout is drawn and written to a final postscript file. The next subsections describe these three steps in more detail.

6.5.1 Generating the MIP

Before we can actually generate a mixed-integer program as described in the previous sections of this chapter, we need a way to input the metro graph and represent it within the Java program. Then, we can loop through vertices and edges and create the constraints and the objective function according to the model of the previous sections.

The input to our program is given in the GraphML format [gra, BEH+01], an extendible and comprehensive XML-based file format to describe the structural properties of a graph. Vertices of the graph have a unique identifier, x- and y-coordinates and a station name used for labeling. Edges of the graph contain, apart from an identifier, a set of Boolean attributes that describe their line affiliations. For the line affiliations we first declare all metro lines of the input system with an identifier, a name and a line color. Each edge that belongs to a certain line contains a *true* entry for the attribute of that line. Using these Boolean attributes to model multiple lines along one edge avoids the need for a multigraph with explicit parallel edges. It is only in the final drawing that we draw multiple parallel lines for such an edge.

To represent the metro graph in the program we adapted the existing graph data structure of the free software library Java Universal Network/Graph Framework (JUNG) [jun]. Classes for metro vertices and metro edges were created as subclasses of the JUNG vertices and edges. These subclasses contain fields to store additional information such as coordinates, sectors for the relative position, circular lists for the embedding etc.

After the input file is read and the data structure for the graph is created, some preprocessing takes place. The graph is checked for edge intersections (assuming that edges are drawn as straight lines in the input) and dummy vertices are inserted if necessary. The input embedding is determined, also assuming straight line edges. Then, optionally, degree-2 vertices are collapsed as described in Section 6.3.1. Depending on the planarity heuristics to apply, an additional data structure for the

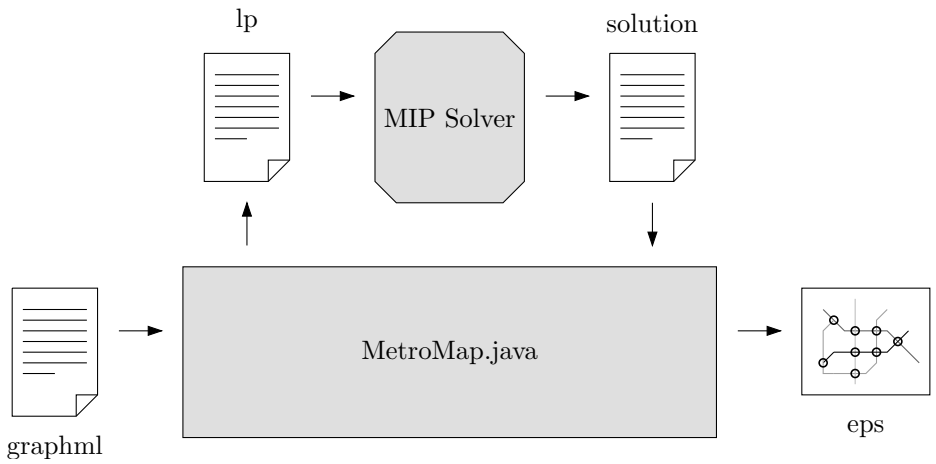


Figure 6.5: Diagram that shows the process of creating a metro map. First the metro graph is read in, the linear program is created, and written to a file. Second, a MIP solver is used to generate a solution file, which is in turn read by our program to finally produce the metro-map layout as an encapsulated postscript file.

faces of the graph or dummy edges for the convex hull are computed. Then, the graph is ready to be transformed into the MIP and the parameters for the weighting of the individual objective functions are requested from the user.

To represent the MIP we designed classes for the entities that appear in a MIP: variables, summands, constraints, objective function and the full mixed-integer program. Step-by-step the MIP is created as detailed in the beginning of this chapter. Variables have a type (integer, binary or real) and lower and upper bounds. In combination with a scalar factor a variable builds a summand. Constraints consist of a list of summands, a constraint type ($\leq$, $=$, $\geq$) and a constant right-hand side. The objective function is simply a list of summands. Once the whole MIP has been constructed, it can be written to a plain text file in the row-oriented LP format (see [ILO] or Appendix C of [glp]), an established standard format for linear programs.

6.5.2 Optimizing the MIP

We decided to accept the inconvenient intermediate steps of creating a text file that contains the MIP, solving it with some optimizer and reading in the solution again in order to be independent from commercial MIP optimizers. Most of these optimizers do offer a programming interface but then it would be more difficult to change to a different optimizer. Moreover, the program would be unusable without a license for a specific optimizer.

We have applied both ILOG CPLEX 9.0 and Dash XpressMP for solving our MIPs. Without closer investigating the reasons we found that CPLEX generates better and faster results for our MIPs so that all the results in the next chapter were produced with CPLEX. For most of the problems the optimizer did not succeed in finding an optimal integer solution within the given time. However, (suboptimal) feasible solutions of high quality could be generated within an acceptable amount of time, see the results in the next chapter. In a solution of the MIP the only variables of interest are the vertex coordinates. These are written to a text file and read into the Java program to update the vertex coordinates.

6.5.3 Graphical Output

Having updated the vertex coordinates the resulting metro-map layout can now be drawn. The graphical output of our Java program is an Encapsulated PostScript (eps) file. PostScript [Ado99] itself is a stack-oriented programming language for page descriptions. As such it is tailored to output textual and graphical data on printers and other devices. Encapsulated PostScript is more restricted than PostScript to make it easier to embed within other documents. Since a metro map drawing consists mainly of colored lines, some circles and possibly text labels the PostScript language is perfectly suited for our purpose. Moreover, PostScript output is freely scalable and of high quality.

For the output the size of the drawing area is computed and scaled such that the maximum side length of the layout equals 500 points. The color of the metro lines is set according to the input color in the metro graph file. If edges belong to multiple lines these lines are drawn parallel to each other and in their respective color. Therefore, the total width of edges that belong to five or more lines gets quite high and they might not look very pleasing and clear in the drawing. Interchange stations are drawn as black circles with a white center and all degree-2 vertices as simple black tick marks. For edges representing collapsed vertices these ticks are placed equidistantly along the edge. Termini of the lines are shown as thicker tick marks.

Chapter 7

Experimental Results

In this chapter we will show how our MIP method performs on six selected real-world examples. We compare our layout with the respective official metro maps designed by professional cartographers. One of the examples has been used in both previous works on metro-map layout [HMN05, SR04] and we compare our result against theirs. The examples are of increasing complexity from a simple system like the Montreal Metro to the highly sophisticated London Underground system. The last example, the S-Bahn RheinNeckar, is a rather simple network but we included station labels in the MIP to show how our method deals with labeled metro maps.

For generating the results in this chapter we used the MIP optimizer ILOG CPLEX 9.0 running on a Power3-II processor with 375 MHz and the operating system AIX 5.1. Today, this is a rather slow machine but it is the only system with a CPLEX license that was accessible to us. Hence, the given running times are likely to be much shorter when using the optimizer on up-to-date hardware.

7.1 Montreal

Our first example is the metro system of Montreal. As indicated in Table 7.1 it consists of four metro lines serving 65 stations (data from 2002). The graph has a simple structure with only three faces. We applied the 3-link heuristic to reduce the graph size to 26 vertices and 27 edges. The resulting MIP size is shown in Table 7.2. Each column gives the MIP size for one of the six planarity options (see Section 6.3.2) from planarity constraints for all non-incident edge pairs to no edge pairs at all. The rows show the number of variables and constraints in the MIP as well as the number of edge pairs forced to be non-intersecting by each of the planarity heuristics. Recall that each such pair gives rise to 33 constraints and 8 variables. Observe that the planarity setting *faces*, which guarantees a planar

	original	3-link paths	collapse all
vertices	65	26	11
edges	66	27	12
faces		3	
lines		4	

Table 7.1: Size of the Montreal metro graph before and after collapsing vertices.

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
variables	2951	2687	1495	1991	999	471
constraints	11241	10152	5235	7281	3189	1011
edge pairs	310	277	128	190	66	0

Table 7.2: Total number of variables, constraints and enforced non-intersecting edge pairs of the Montreal MIP for six different planarity settings.

drawing, gives rise to about 90 percent of the MIP constraints and more than 80 percent of the variables. Applying the heuristics significantly decreases the number of edge pairs forced to be disjoint and thus the MIP size itself.

Figure 7.1 shows the geographic layout of the Montreal metro system and Figure 7.2 displays the result that our method produced from the geographic input. The weights in the objective function were $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 2, 2)$. We could skip the planarity constraints here (last column of Table 7.2) since the result was planar anyway. The solution shown was computed within 31 seconds. It is a clear octilinear layout using many diagonals because they best approximate the relative positions of the input. Only few bends are used and all have the preferred obtuse 135° angle. Stations are spaced very uniformly along the edges. In comparison to the official map (see Figure 7.3 [Soc]) our layout is of similar quality with respect to the metro map esthetics. The official map has a different appearance due to the inclusion of the simplified geographic shape of the Saint Lawrence River which puts some restrictions on the layout. Especially the green line is meandering towards both its ends which is necessary to not lead into the river. In spite of these extra bends the official layout is still clear because the network is small and it involves only four lines. However, our layout needs less bends and could be applied if a metro map without geographic background is desired.

In Figure 7.4 we show another layout of the Montreal metro. For this result we used a different weight λ_{relpos} for the relative-position to demonstrate how this affects the drawing. So we changed the parameter from $\lambda_{\text{relpos}} = 2$ to $\lambda_{\text{relpos}} = 4$. Now, the layout may now contain more bends in favor of better preserving the relative positions of the geographic input layout. And indeed, the shape of the lines resembles their geographic shape more than in the previous layout of Figure 7.2. However, Figure 7.2 contains less line bends, as expected, and looks clearer and more balanced than our second layout. This observation suggests that the bend criterion is more important than the relative position unless there are reasons that require to retain a stronger idea of the input geometry.

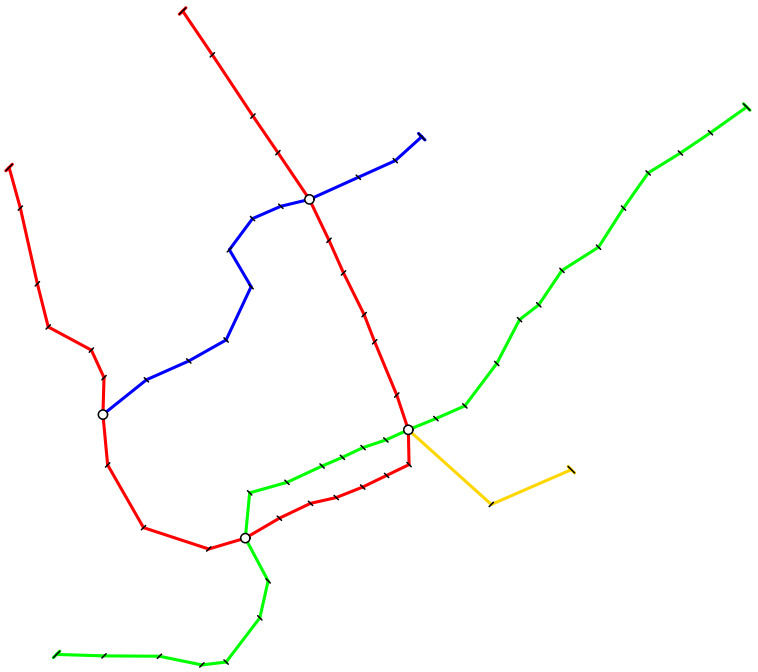


Figure 7.1: Original geographic layout of the Montreal metro.

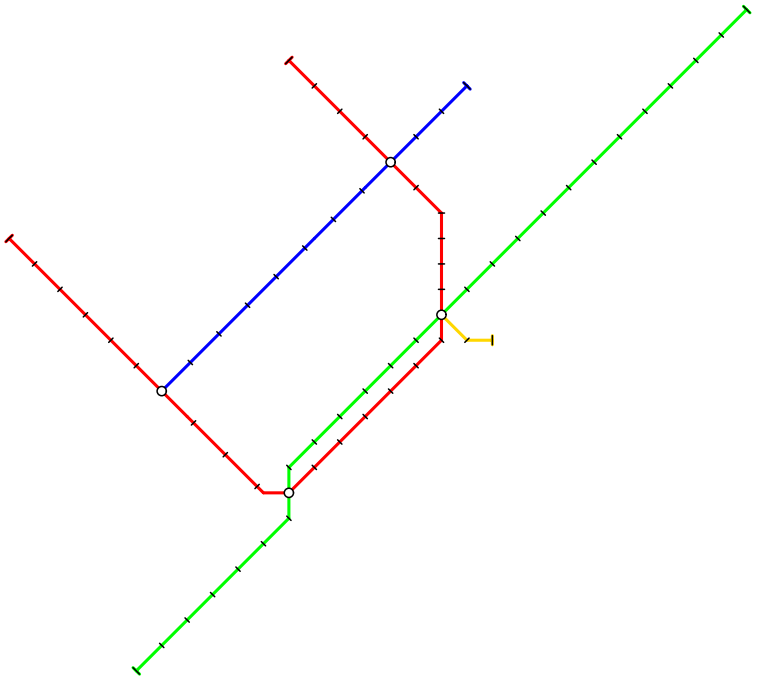


Figure 7.2: Final layout of the Montreal metro using our method.

Figure 7.3: Official map of the Montreal metro.

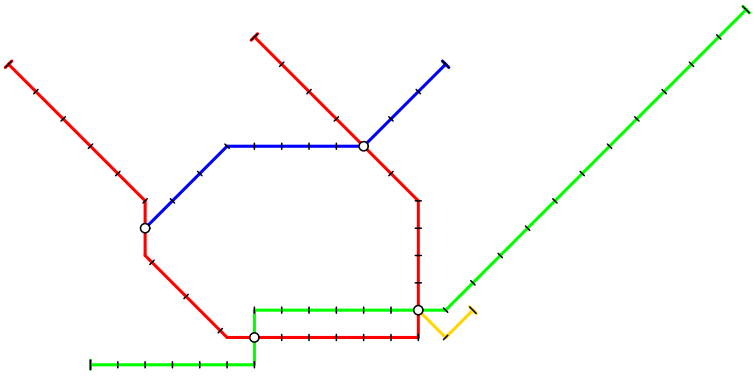


Figure 7.4: Layout of the Montreal metro emphasizing the soft constraint for relative position.

7.2 Vienna

The metro system of Vienna consists of five lines with 90 stations and 96 edges, see Table 7.3. With the 3-link heuristic the size could be reduced to 44 vertices and 50 edges. As can be seen in the original layout in Figure 7.5, the city center is more connected by metro lines than in the previous example. There are ten interchange stations between the different lines and the metro graph has eight faces.

	original	3-link paths	collapse all
vertices	90	44	19
edges	96	50	25
faces	8		
lines	5		

Table 7.3: Size of the Vienna metro graph before and after collapsing vertices.

The resulting MIP sizes for Vienna are given in Table 7.4. Even when using the face method, the planarity constraints amount for about 90 percent of the MIP size. As before we could omit the planarity constraints completely so that the last column gives the actual MIP size. Figure 7.6 shows the result that our method produced from the geographic input in Figure 7.5. The weights in the objective function were $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 2, 2)$. The solution shown was obtained within only 26 seconds.

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
variables	9960	6048	2872	4176	1800	872
constraints	39363	23226	10125	15504	5703	1875
edge pairs	1136	647	250	413	116	0

Table 7.4: Total number of variables, constraints and enforced planar edge pairs of the Vienna MIP for six different planarity settings.

Our layout succeeds in schematizing the input in a clean way. The lines show few bends and only one of them has a right angle instead of 135°. This happens where the green and orange lines meet and is almost unavoidable when looking at the geographical situation at this station. Lines pass interchange stations as straight as possible. The left end of the orange line could be drawn with one bend less but the layout favors the relative position here and approximates the geographic run of the line. The total edge length is small and stations are spaced equidistantly. The layout also shows some nice symmetries that give it a distinct look. It is hard to compare our map directly with the official map[1] (see Figure 7.7 [Wie]) because the official map has quite different dimensions to make it fit into the metro cars. Obviously, that format requires more bends and less geographic accuracy.

[1] The official map does not show the extension of the purple line which is under construction.

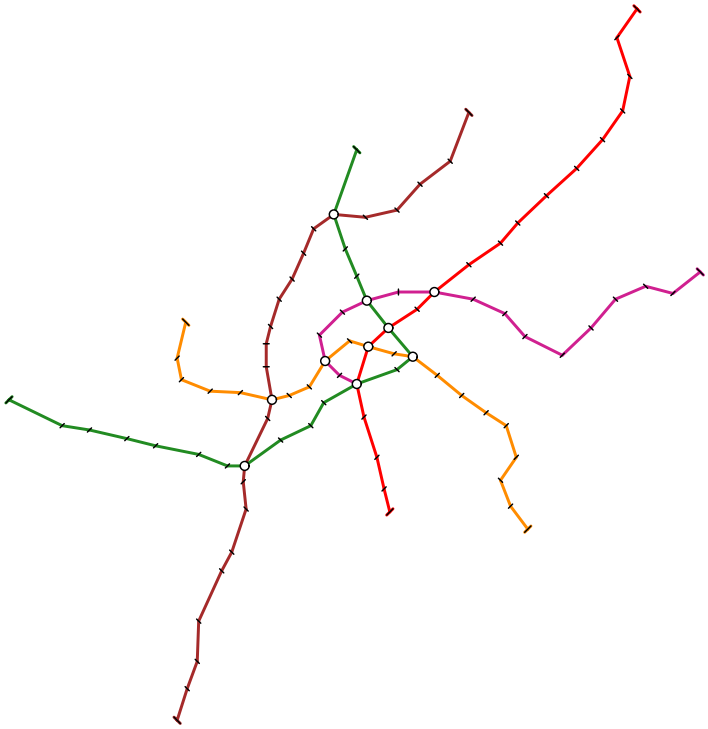


Figure 7.5: Original geographic layout of the Vienna metro.

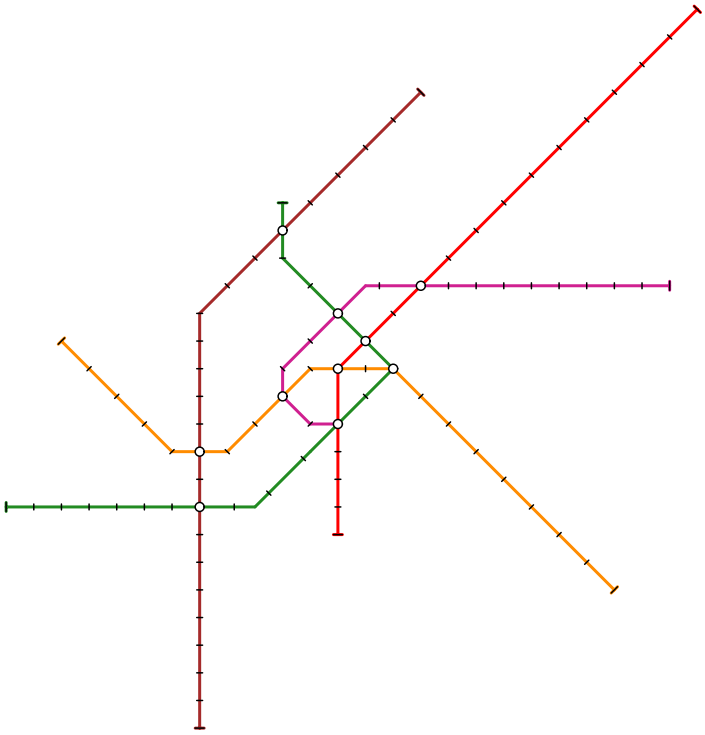


Figure 7.6: Final layout of the Vienna metro using our method.

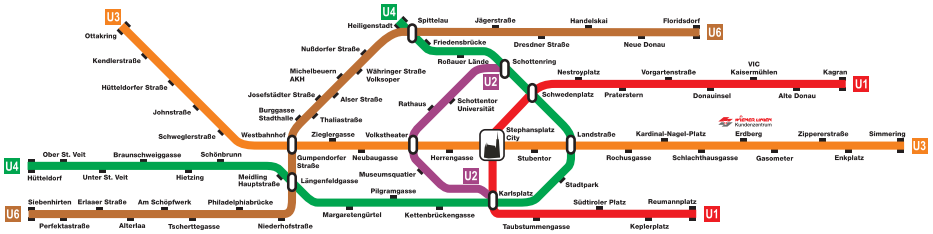


Figure 7.7: Official Map of the Vienna metro.

7.3 Karlsruhe

Our third example is the tram system of Karlsruhe. Although not being a metro system in a strict sense because the trains run along regular roads and not in tunnels, we included the network as an example.[2] The complete network is quite large and expands up to 80 km into the surroundings. Hence, we clipped the network to represent only the area of Karlsruhe itself. As shown in Table 7.5, the graph still has 126 vertices, 132 edges, and ten metro lines. Some edges are shared by up to eight lines. The clipped original geographic layout is displayed in Figure 7.8. Due to Karlsruhe's regular fan-shaped road pattern in the downtown area, the central part has already a quite schematic appearance. Starting from this central part, the network branches into the surroundings in many directions. The long purple branch in the lower left part of that layout is caused by a train that uses the regular long-distance rails and hence stations are much further apart than in the rest of the network. We applied the 3-link heuristic to reduce the graph size to 70 vertices and 76 edges.

	original	3-link paths	collapse all
vertices	126	70	35
edges	132	76	41
faces		8	
lines		10	

Table 7.5: Size of the Karlsruhe metro graph before and after collapsing vertices.

This reduced graph led to the MIP sizes shown in Table 7.6. The values are large in comparison to the previous examples and since the network has many pendant edges, the pendant-edge heuristic still forces a large number of edge pairs to be disjoint. In connection with the convex-hull heuristic the numbers become more tractable but we were fortunate and could draw a planar layout in spite of skipping the planarity condition completely. Figure 7.9 shows the result using the weights $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 4, 3)$. This solution was obtained after 200 seconds.

Our layout for Karlsruhe succeeds in finding a nice and symmetric way to display the downtown area. There are some right angles but they do not disturb the graph layout itself. The long purple branch in the southern part of the original layout has been shortened in our layout which might not be desired in this case because the two stations involved have a much larger distance in reality. We could avoid this by setting the minimum length of this edge accordingly. One serious drawback of our layout, which is however caused by the geometry and structure of the input, is

[2] After all it is a network that people at Universität Karlsruhe should be familiar with.

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
variables	23330	17210	6978	10858	4186	1434
constraints	93808	68563	26356	42361	14839	3487
edge pairs	2737	1972	693	1178	344	0

Table 7.6: Total number of variables, constraints and enforced planar edge pairs of the Karlsruhe MIP for six different planarity settings.

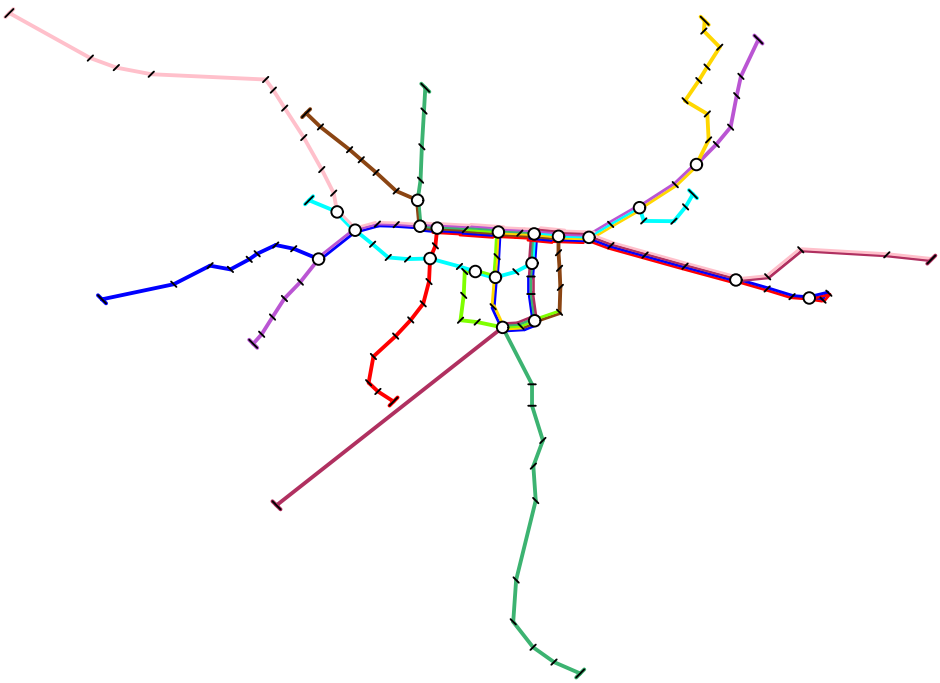


Figure 7.8: Original geographic layout of the Karlsruhe tram system.

that it is difficult to see where the vertical lines in the central part continue when hitting the main horizontal line. In the worst case there are eight lines sharing one edge and the individual line colors are hard to distinguish. One needs to look very closely to see whether a line from the south goes left or right at these points. In the official layout (see Figure 7.10 from 1997[3] [Kar]) this problem is solved by explicitly drawing multi-edges for example on the main horizontal line. To that end our algorithm would need to model stations not as points but as rectangles. However, a cartographer could still take our drawing of the simple graph as basis to insert multiedges and rectangular stations where applicable. In that case our layout and the official map would be very similar.

[3]We use this old official map because newer maps always show the complete network and would be harder to compare with our layout. Note that the light-green metro line in Figure 7.9 is not yet contained in the 1997 map.

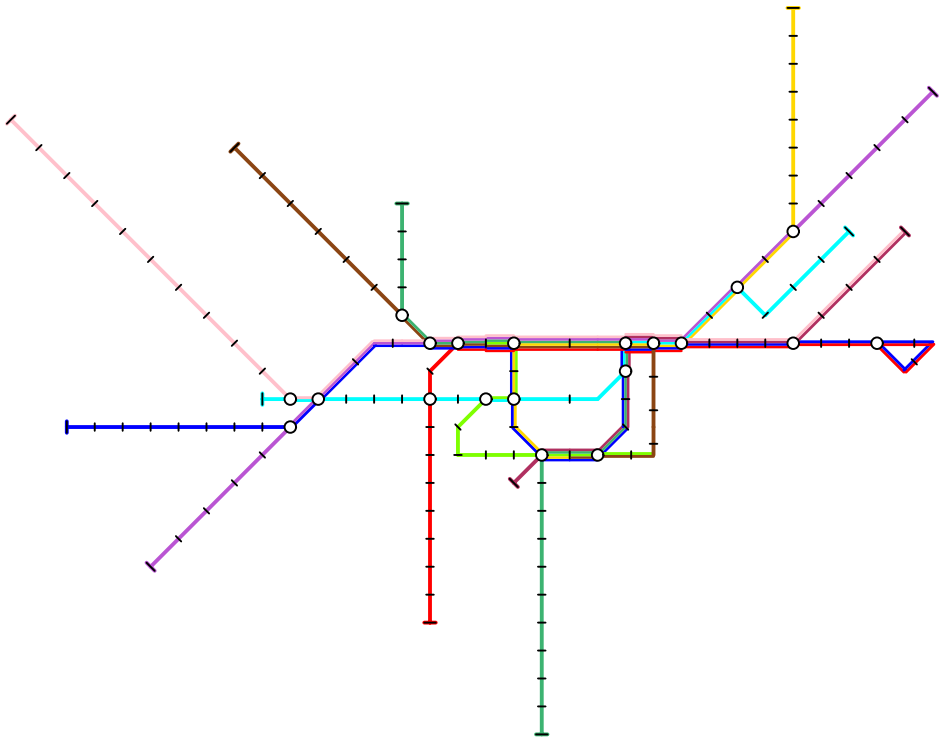


Figure 7.9: Final layout of the Karlsruhe tram system using our method.

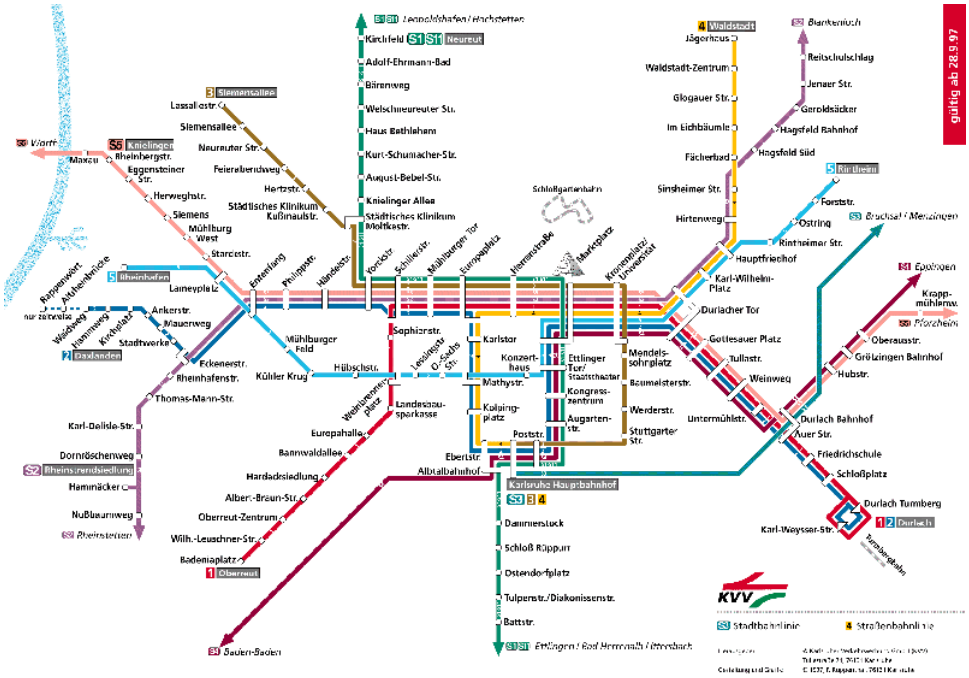


Figure 7.10: Official Map of the Karlsruhe metro system.

7.4 Sydney

The CityRail system in Sydney (restricted to the suburban part) is an example of a larger network with ten lines, 174 stations and 183 edges, among them several multiple edges. Collapsing vertices with the 3-link heuristic leads to a reduced graph of 62 vertices and 71 edges, see Table 7.7. The original layout is displayed in Figure 7.11. One can see that the downtown area at the right-hand side is very condensed in the geographical drawing. Moreover, there is one edge intersection in that area, not visible at this scale, that requires to add a dummy vertex.

	original	3-link paths	collapse all
vertices	174	62	31
edges	183	71	40
faces		11	
lines		10	

Table 7.7: Size of the Sydney metro graph before and after collapsing vertices.

The MIP characteristics for the reduced graph are given in Table 7.8. In terms of vertices and edges the graph of Sydney is of similar size as the one of Karlsruhe. However, applying the planarity heuristics decreases the MIP size more than in the previous example. For Sydney we had to use the combined convex-hull & pendant-edges heuristic (second last column in Table 7.8) because of a persistent edge intersection. This heuristic and the parameter setting $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 5, 5)$ yielded the layout in Figure 7.12 within 22 minutes.

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
variables	20329	11545	5921	3873	2105	1329
constraints	81416	45182	21983	13535	6242	3041
edge pairs	2375	1277	574	318	97	0

Table 7.8: Total number of variables, constraints and enforced planar edge pairs of the Sydney MIP for six different planarity settings.

Our drawing is a clear octilinear layout of the Sydney network. As Sydney has several shared edges there is one thick edge on the right-hand side of our layout which is made up of six lines. However, it is less difficult to distinguish the different line continuations than in the previous example. In comparison to the official layout (see Figure 7.13 [Syd]) there are two aspects that attract attention. The loop in the east of the drawings is more symmetric and emphasized in the manually designed metro map. Further, the green line in the south-eastern part of the system stays quite close to its neighboring parallel line in our drawing. This effect is caused by minimizing the total edge length since the shortest possibility to draw that part of the green line is the one that is shown in our drawing. The official map puts these lines at a larger distance.

Sydney has been used as an example before [HMN05, SR04] (see Section 3.2) and hence we can compare our result against these earlier results. Figure 7.14 is taken from Hong et al. [HMN05] and shows their layout using the most refined of their spring embedder methods. Originally they draw a slightly larger network including some intercity lines that extend the suburban network. However, these extensions should not influence the layout of the central part of the network. For

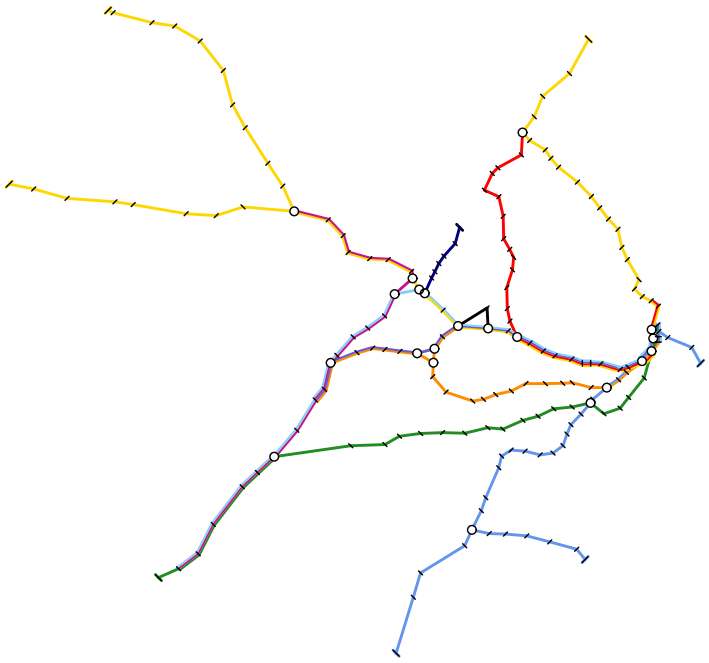


Figure 7.11: Original geographic layout of the Sydney CityRail system.

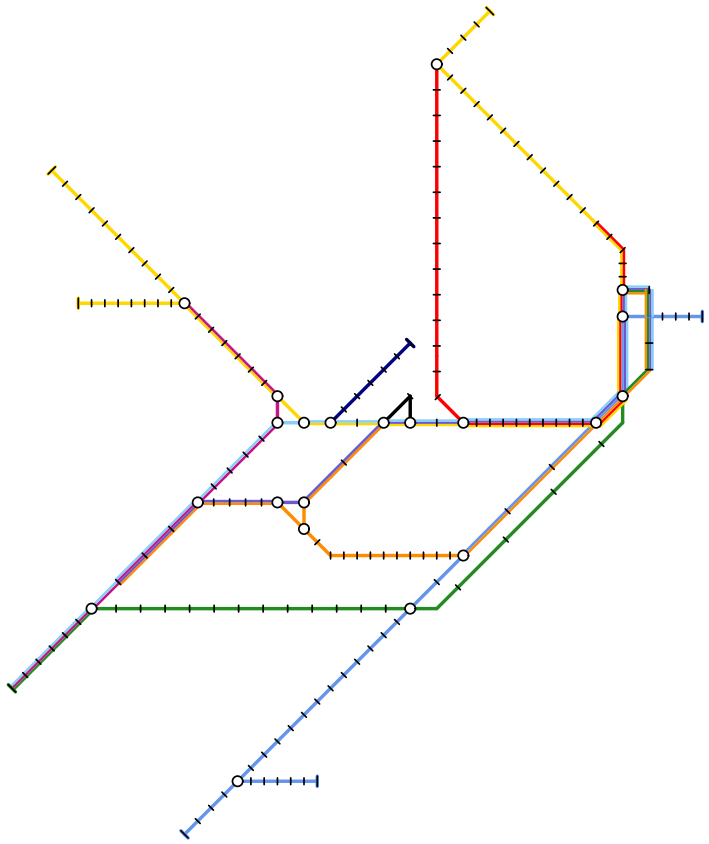


Figure 7.12: Final layout of the Sydney CityRail system using our method.

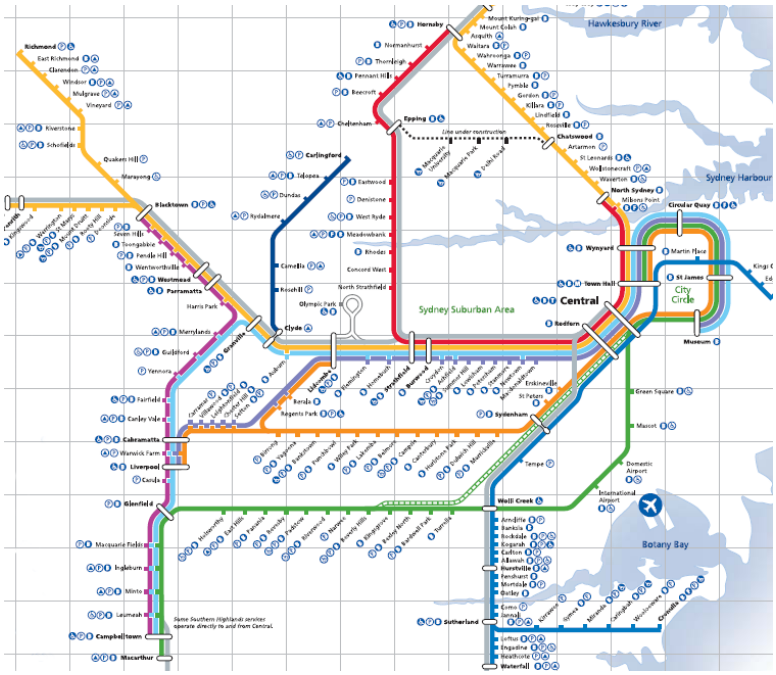


Figure 7.13: Official Map of the Sydney CityRail system.

ease of comparison we clipped the lines appropriately in Figure 7.14. Apart from the fact that Hong et al. show station labels, one can observe that edges are not strictly octilinear and that avoiding bends along lines is not a goal of their method. In addition, there is a large variance of the edge lengths.

Stott and Rodgers [SR04] apply a multicriteria optimization algorithm to produce their metro-map layouts. Figure 7.15 shows their result when collapsing all degree-2 vertices before actually drawing the network. There are two non-horizontal edges that obviously violate octilinearity, which is the most important drawback of this layout. Figure 7.16 displays the result of the same method without prior vertex collapses. It now shows an almost octilinear layout with the exception of one edge on the left side. Both non-octilinear edges from Figure 7.15 are successfully made octilinear by introducing bends. On the other hand, some unnecessary bends on peripheral lines could not be removed by the local multicriteria optimization technique. Interestingly enough, it seems that they do not planarize the original embedding: their layout of the Sydney CityRail is not quite topologically equivalent to the corresponding original embedding (see the tail within the eastern loop of the network).

The method of Stott and Rodgers produces layouts that are more similar to real-world metro maps than the layouts of Hong et al. But both methods have serious drawbacks. Neither of them guarantees octilinearity. Hong et al. do not consider line bends in their method which seems to be an important esthetic for metro maps. Stott and Rodgers do minimize bends within their objective function. However, the algorithm gets easily trapped in local minima and their layouts show defects that a professional designer would correct immediately.

Our method overcomes the limitations of the previous results. Most importantly, there are no exceptions to octilinearity. A further improvement is that the global optimization of a mixed integer program avoids the problems of local minima in [SR04]. In contrast to Hong et al. we actively minimize the number of line bends in

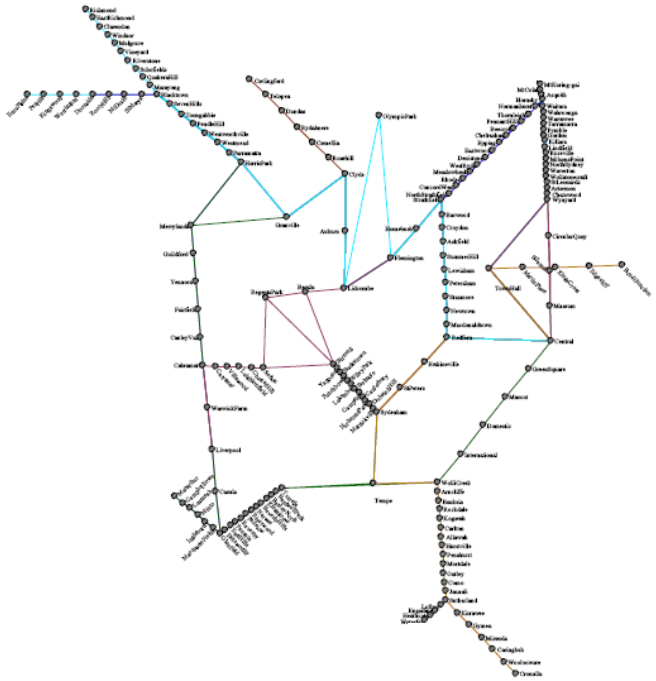


Figure 7.14: Layout of the Sydney CityRail system by Hong et al. (clipping of Figure 7(b) in [HMN05])

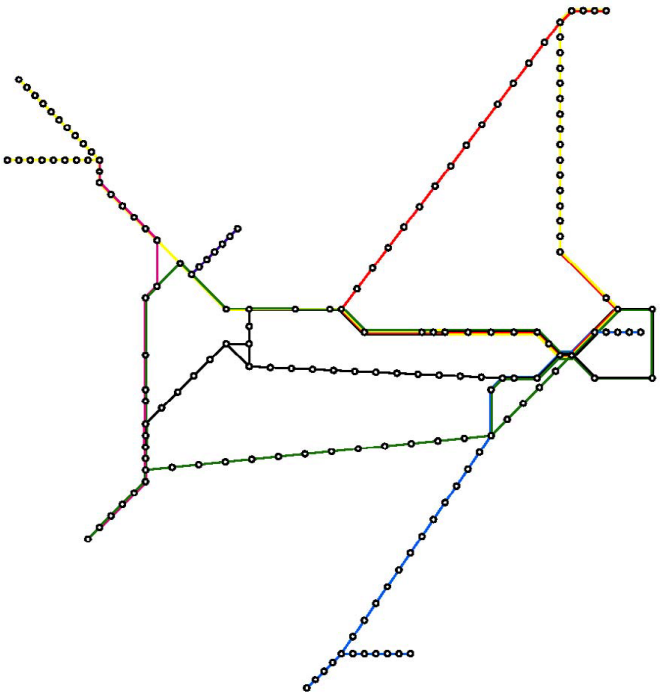


Figure 7.15: Layout of the Sydney CityRail system by Stott and Rodgers (Figure 14 in [SR04]) using a reduced graph.

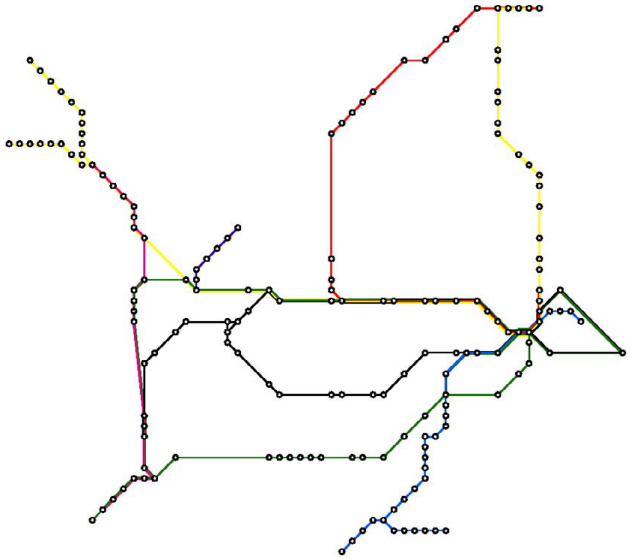


Figure 7.16: Layout of the Sydney CityRail system by Stott and Rodgers (Figure 15 in [SR04]) using the original graph.

the layout and try to maintain the overall geography using the concept of relative positions. Comparing the results of all three methods visually, we claim that the layout of our method is most similar and of comparable quality to a manually designed metro map.

The main disadvantage of our method is its running time. While we needed 22 minutes to produce our Sydney map, Hong et al. computed the layout in Figure 7.14 within only 7.6 seconds. Stott and Rodgers needed 4 minutes for a Sydney map using a contracted input graph (Figure 7.15) and about 28 minutes for the uncontracted graph (Figure 7.16). However, the experiments were carried out on very different machines and from a practical point of view it is worth spending more running time for getting significantly better results as long as maps do not have to be drawn interactively. Moreover, our method usually generates feasible but suboptimal solutions within a few seconds.

7.5 London

Certainly one of the most complex metro systems is the London Underground. It is also the oldest metro system of the world with the first line opened in 1863. Nowadays, for many Londoners the Tube Map has become the mental map of the city rather than London's geography. Hence, only careful and small changes to the layout will be accepted by the public in reality. Nevertheless, automatically drawing a metro network of that size and complexity is an interesting challenge for our method. As Table 7.9 shows, the original graph has 309 vertices (among them a few dummy vertices representing edge intersections) and 361 edges used by 13 metro lines and forming 54 faces. The central part of the network is highly connected as can be seen in the geographic layout in Figure 7.17. Therefore applying the 3-link heuristic only reduces the size to 181 vertices and 233 edges.

	original	3-link paths	collapse all
vertices	309	181	99
edges	361	233	151
faces	54		
lines	13		

Table 7.9: Size of the London metro graph before and after collapsing vertices.

We used this reduced graph to create the MIP where the actual numbers of variables and constraints for the different planarity settings are shown in Table 7.10. Looking at the first two columns underlines how large the network really is: 26529 edge pairs considering all non-incident edges and still 5777 edge pairs using the face structure of the input embedding are required to be non-intersecting. Considering the first column more than 99 percent of the constraints are required for the planarity of the drawing. These numbers could be reduced drastically with the planarity heuristics. We had to use the combined convex-hull & pendant-edges heuristic (second last column in Table 7.10) to create a planar drawing. This heuristic and the parameter setting $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 4, 4)$ yielded the layout in Figure 7.18 within 4.5 hours.

Obviously not as balanced and sophisticated as the official Tube Map in Figure 7.19 [Traa], our layout in Figure 7.18 shows an octilinear metro map of mixed quality. A positive point is for example that the central part of the red line is drawn horizontally. In most interchanges lines pass straightly or having at most an angle of 135°. However, at some points the geography enforces acute 45°-angles between two adjacent edges of one line. The shape of the yellow Circle line is quite close to the striking flask shape in the official map. But especially towards the boundary of our drawing the lengths between adjacent stations get rather small in comparison to the inter-station distance in the center of the drawing. And for example the two

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
variables	216173	50157	19845	18589	7981	3941
constraints	884231	199415	74378	69197	25439	8774
edge pairs	26529	5777	1988	1831	505	0

Table 7.10: Total number of variables, constraints and enforced planar edge pairs of the London MIP for six different planarity settings.

Figure 7.17: Original geographic layout of the London Underground.

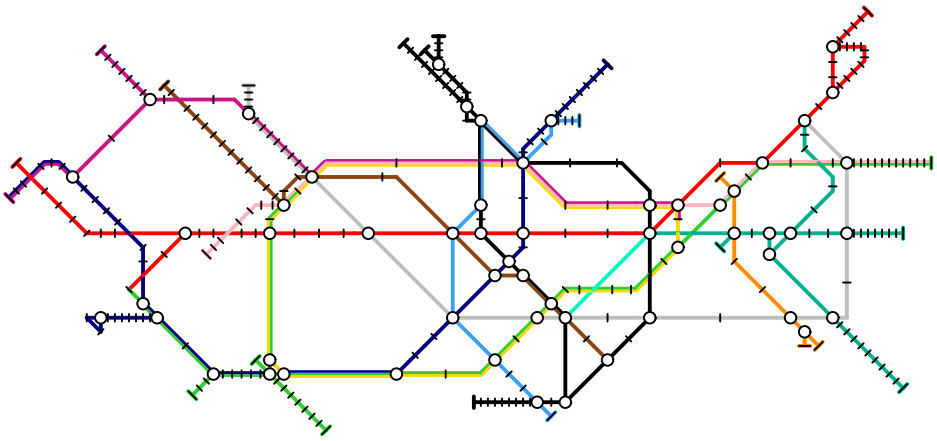


Figure 7.18: Final layout of the London Underground using our method.

branches of the black line in the top middle are drawn very close together which is misleading with regard to their true distance (see Figure 7.17). However, considering the complexity of the London network we were fortunate enough to be able to produce a drawing at all—and the result could look much worse. This shows that our method is indeed capable of drawing large real-world metro maps. Our layout could certainly be used as a basis for manual improvements by a graphic designer.

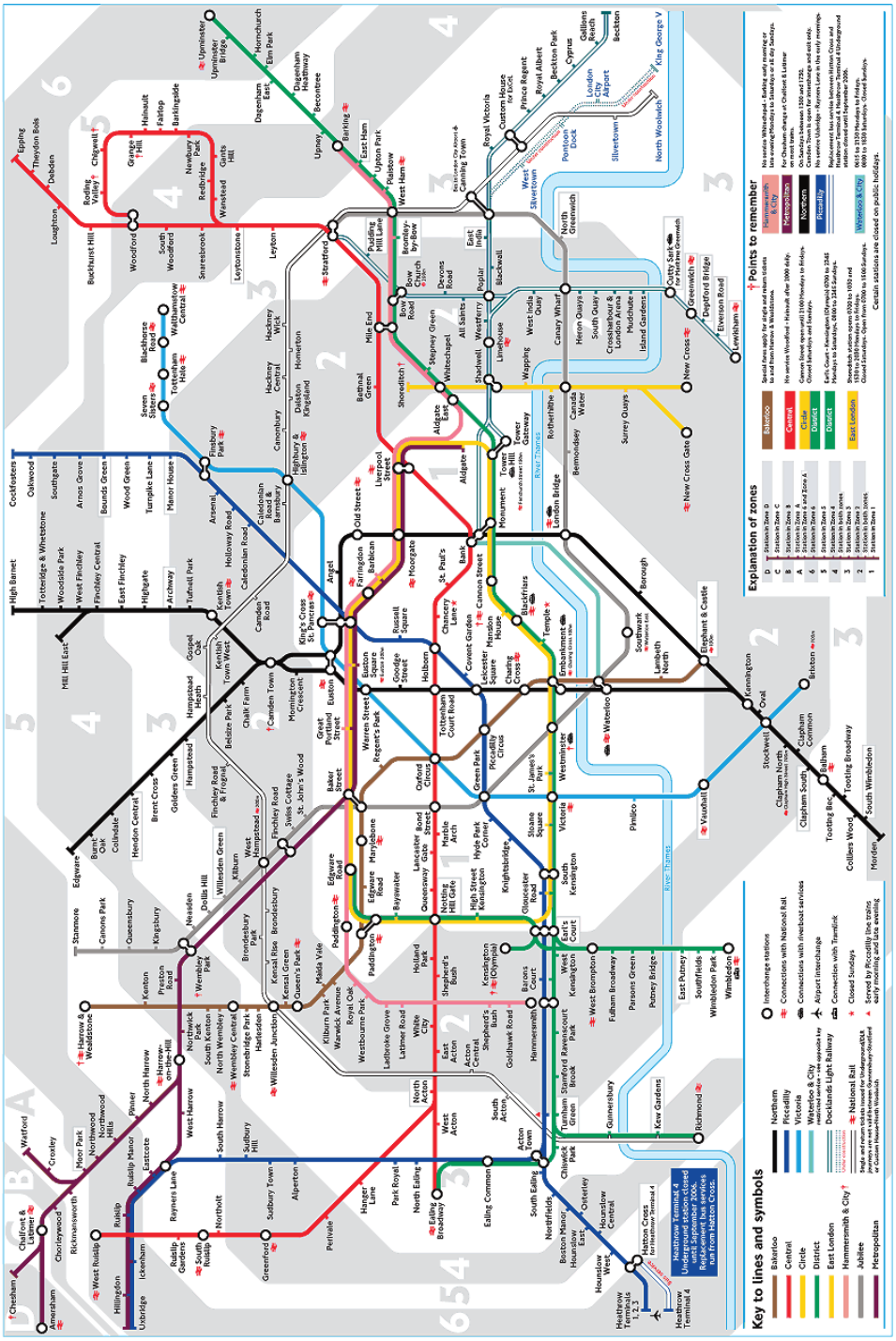


Figure 7.19: Official map of the London Underground.

7.6 S-Bahn RheinNeckar

The S-Bahn RheinNeckar [S-B] is no metro system in the strict sense. The network comprises the railway tracks that are used by the regional train service S-Bahn RheinNeckar in the area around Heidelberg, Mannheim and Karlsruhe (see Figure 7.20 for the geographic layout). It spans 150 km in the east-west direction. However, it is still a public transport network that might benefit from having a nice metro-map layout. Due to its simple structure we also experimented with labeled drawings for this network.

The size of the input graph and the corresponding reduced graphs is given in Table 7.11. Originally it has 108 vertices and 111 edges. Six lines belong to the network, two of them operated by a different company. In the unlabeled case we decided to collapse all degree-2 vertices because links proved to be unnecessary here. The remaining graph has 42 vertices and 45 edges. For the labeled graph we had to collapse all degree-2 vertices by construction anyway but modeling the labels with parallelograms adds new vertices and edges such that the resulting labeled metro graph has 127 vertices and 144 edges. The number of faces increases from 5 to 19 since the parallelograms represent new faces of the graph.

	original	3-link paths	collapse all	with labels
vertices	108	74	42	127
edges	111	77	45	144
faces	5			19
lines	6			

Table 7.11: Size of the RheinNeckar metro graph before and after collapsing vertices.

Concerning the MIP sizes we have to distinguish between the unlabeled and labeled case again. Table 7.12 shows the numbers for both cases. The unlabeled graph is quite small and so are the MIP sizes in comparison to previous examples. For the labeled graph the numbers are much higher and we necessarily need to include at least some planarity constraints that restrict labels to not overlap with other labels or edges.

First, we describe the unlabeled drawing displayed in Figure 7.21. We used the parameter setting $(\lambda_{\text{length}}, \lambda_{\text{bends}}, \lambda_{\text{relpos}}) = (1, 2, 2)$ and could solve the MIP

	all pairs	faces	convex hull	pendant edges	convex hull & pendant edges	none
unlabeled						
variables	7961	7129	2617	5553	1929	697
constraints	31654	28222	9610	21721	6772	1690
edge pairs	908	804	240	607	154	0
labeled						
variables	81477	43781	34605	13813	11501	1309
constraints	334034	178538	140687	54920	45383	3341
edge pairs	10021	5309	4162	1563	1274	0

Table 7.12: Total number of variables, constraints and enforced planar edge pairs of the RheinNeckar MIP, either unlabeled or labeled, for six different planarity settings.

to optimality in 43 seconds. This was in fact the only example where we found a provably optimal solution. It is a very simple and clear drawing of the graph when compared to the input network in Figure 7.20. The meandering of the east-west line was completely removed by straightening it to a horizontal line. The southern part of the network is drawn very clearly as well. Note that the gray edges that are attached to the main network indicate train services that connect to the S-Bahn RheinNeckar network.

The official map in Figure 7.22 [S-B] has a layout that does not satisfy our metro-map esthetics well. It stays very close to the geographic layout which results in many (unnecessary) bends along the lines. Some of the edges even make sharp 90° turns that could be avoided by simply drawing a straight-line edge instead. The official layout also includes two schematized rivers but these are the only geographic features in the drawing and could still be simplified further. Since the area covered by the network is very large and comprises several cities there is not really a strong mental map of the region which needs to be strictly preserved. The system operator has decided to design a schematic octilinear and not a geographic network map and therefore it is not comprehensible to us why they stick so closely to the geometry at the cost of an overly complex diagram. We claim that our simple layout in Figure 7.21 serves the purpose of a metro map, namely to act as a navigational aid, much better. We could also include the two rivers, modeled as additional metro lines.

Our labeled drawing is shown in Figure 7.23. The parameters for the objective function were identical to the unlabeled case. However, since the MIP was of much larger size this layout was obtained after 20 hours instead of only 43 seconds for Figure 7.21. To demonstrate how the parallelograms for the labels work we included them in the drawing. They are attached to the edges and shown in light gray. Their dimensions depend on the longest station name. Interchanges are simply labeled by an edge of the correct length. For reasons of size and running time we had to rely on the planarity heuristics that were originally designed for unlabeled drawings with a fixed embedding. But as mentioned in Section 6.4 the labeled metro graph no longer has a fixed embedding because the label boxes can go on either side of their respective edge. Consequently, not all labels could be drawn overlap-free in Figure 7.23. A MIP avoiding this problem by forcing all edge pairs not to intersect would be far too large to solve in practice. Therefore, this drawing should rather be seen as our first attempt to create a labeled metro map and not yet as a veritable competitor for manually-designed labeled metro maps.

In comparison to the unlabeled layout of Figure 7.21 the metro graph in Figure 7.23 itself, i.e. without the labels, cannot achieve the same simplicity. It has some unnecessary bends, especially along the eastern branch, that disrupt the clean horizontal axis formed by that line in Figure 7.21. But our layout in Figure 7.23 seems to be not worse than the layout of the official graph with its meandering lines (see Figure 7.22) either. Concerning the labels, our layout suffers of course from the overlaps that are present. More effort is necessary to improve on this. The official map has no label-label overlaps but some labels do overlap with the rivers or the gray arrows[4] that indicate connections to different networks. Moreover, the official map uses three directions for the text of labels: horizontal and both diagonals. It would add more clarity to the map if at most one diagonal text direction was used in addition to the horizontal text.

[4]It can be argued that this is tolerable for rivers and arrows.

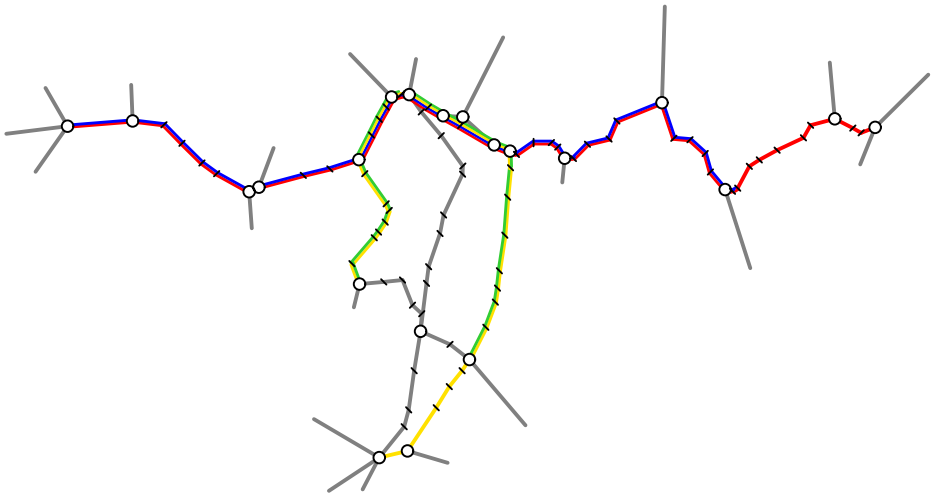


Figure 7.20: Original geographic layout of the S-Bahn RheinNeckar.

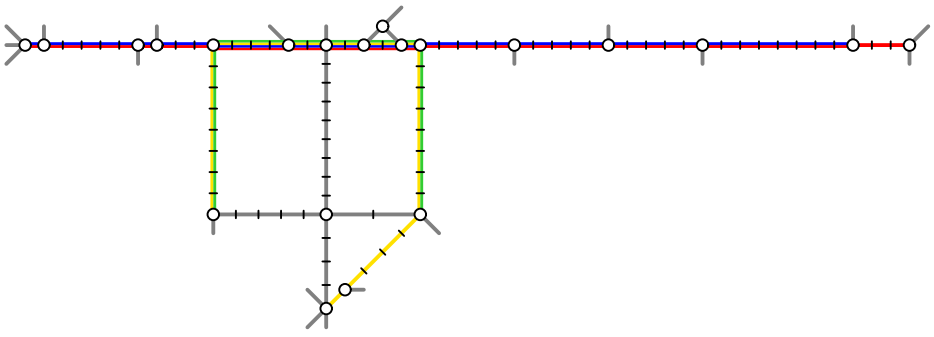


Figure 7.21: Final layout of the S-Bahn RheinNeckar using our method.

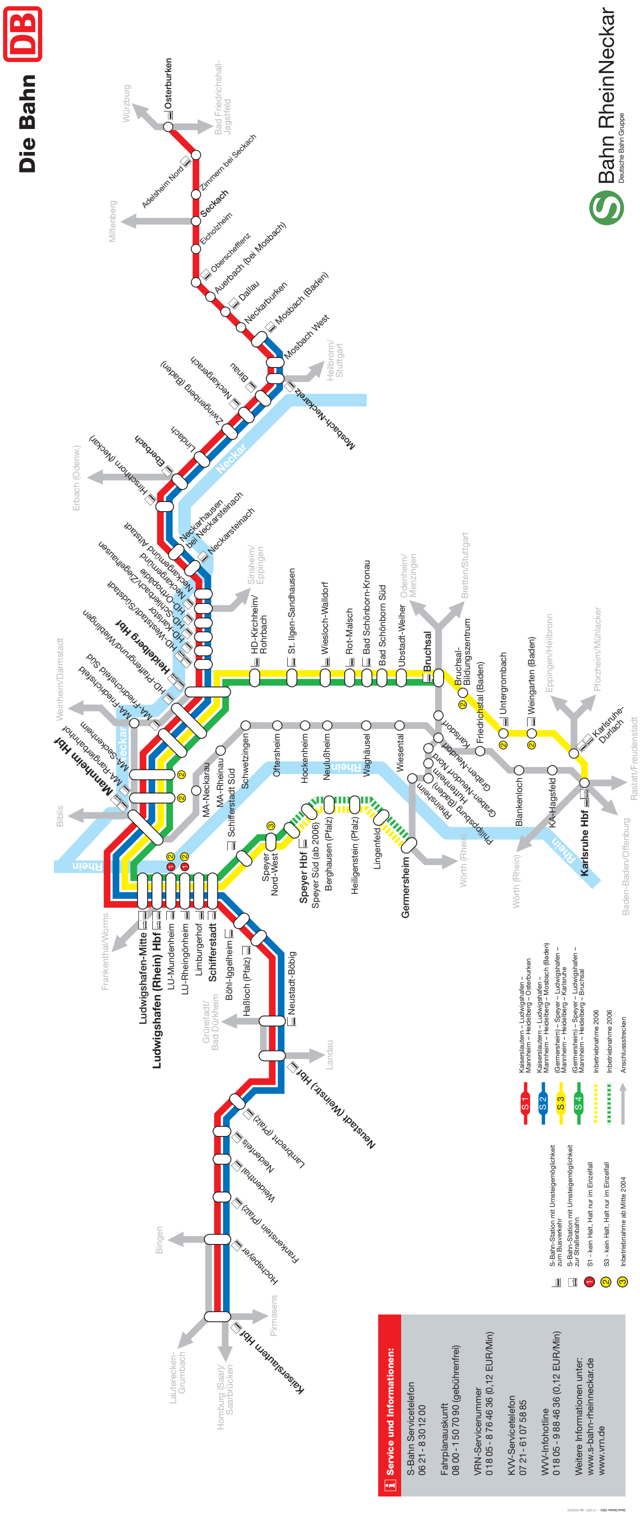


Figure 7.22: Official map of the S-Bahn RheinNeckar.

Figure 7.23: Labeled layout of the S-Bahn RheinNeckar including the label boxes.

Chapter 8

Final Remarks

In this final chapter we conclude our work and point out directions for further work on metro-map layout and, more general, about octilinear drawings. As metro-map layouts represent a very new layout style in graph drawing, many extensions and applications are conceivable.

8.1 Conclusion

In this work we have studied the problem of automatically drawing metro maps. By analysing the layout features of real-world metro maps such as the famous London Underground Map we identified octilinearity and preservation of the embedding as the most important hard constraints for metro maps. Soft constraints to optimize include the number of bends along the metro lines as well as a rough preservation of the input geometry. The theoretical result of this work is the NP-completeness of the MetroMap decision problem, i.e. to decide whether a given graph can be drawn as a metro map. Nonetheless, for solving the layout problem in practice, we have translated the hard and soft constraints into a mixed-integer linear program. To cope with the running times of the MIP optimizer we have also presented and implemented two heuristic speed-up methods. Our experiments with real-world metro networks show that we have succeeded in drawing these networks in a quality comparable to maps drawn by graphic designers. The nature of the MIP optimization process is such that we usually quickly get good intermediate solutions but proving their optimality can take a long time. In practice it might not even be worth waiting for an optimal solution (in terms of the objective function) but use a stable feasible solution instead. Recall that the objective function is just an attempt to mathematically formulate "nice" and hence the optimal solution is not necessarily "nicer" (for a human viewer) than some close-by solutions. Finally, we have also investigated how to model labeled metro maps in the MIP framework. We extended the previous model for unlabeled layouts to comprise labels as special additional faces in the graph. The results show that the model is working, however, our speed-up heuristics do not handle the new situation very well. In order to guarantee overlap-free labels the MIP size is still too large.

8.2 Outlook

There are a few starting points for future work based on this work. A first extension would be to allow for more user interaction during the drawing process. The user could for example wish to force some edges of the graph into a fixed direction. This can be easily incorporated into the current MIP. This would be useful especially when trying to improve a given layout where the nice parts of the drawing should be retained or when a new metro line needs to be inserted into an existing layout. Many other scenarios for user interaction are possible.

Labeling is not solved satisfactorily yet. Either the MIP size becomes too big or labels create overlaps. New speed-up heuristics for labeled metro maps would be useful. Moreover, in the current model labeling cannot be combined with the 3-link heuristic. However, this heuristic usually creates drawings that are visually more appealing than drawings where all degree-2 vertices are collapsed. An alternative idea is to separate generating the graph layout from labeling it. But unlike map labeling where a fixed map is labeled, we could admit that a few changes to the layout, e.g. modifying edge lengths, are made in order to better fit the labels into the given drawing.

In some of the examples we saw that with increasing multiplicity of edges it becomes difficult in our current approach to keep the different colors of the lines apart. Hence, it would be necessary to increase the line thickness and consequently the size of the stations involved, as in the official metro maps. This means that the dimension of vertices and edges has to be integrated into our model or parallel edges must be explicitly modeled. Rectangular vertices could be modeled as in the MIP formulation of Binucci et al. [BDLN05]. This would also make it possible to draw vertices of degree greater than 8.

Octilinear layouts should also be examined with regard to other applications than metro maps. It would also be possible to design a similar model for any kind of technical drawing with a restricted number of orientations as long as the increase in the size of the MIP is acceptable. Depending on the requirements of the application some constraints may be dropped and alternative objective functions could be devised.

Finally, it is definitely worth trying to run the MIP optimizer on up-to-date hardware instead of the old machine that we had available. This might improve especially on the large instances and labeled metro maps. While drawing a public transportation map is a rather static problem where long running times are acceptable, faster algorithms are necessary for application in a more dynamic environment. Thus, in the long run it might be interesting to implement fast heuristics for the problem instead of using the MIP formulation. Alternatively, a custom MIP solver could be developed that is aware of the graph structure and uses this knowledge in solving the MIP.

Bibliography

[Ado99] Adobe Systems Inc. *PostScript Language Reference 3rd edition.* Addison-Wesley, 1999. 54

[AM00] Silvania Avelar and Matthias Müller. Generating topologically correct schematic maps. In *Proc. 9th Int. Symp. on Spatial Data Handling (SDH'00)*, pages 4a.28–4a.35, Beijing, 10–12 August 2000. 14

[Ata99] Mikhail J. Atallah, editor. *Algorithms and Theory of Computation Handbook.* CRC Press, 1999. 80

[BDLN05] Carla Binucci, Walter Didimo, Giuseppe Liotta, and Maddalena Nonato. Orthogonal drawings of graphs with vertex and edge labels. *Computational Geometry: Theory and Applications*, 2005. To appear. 16, 17, 50, 78

[BEH+01] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall. Graphml progress report: Structural layer proposal. In *Proc. 9th Intl. Symp. Graph Drawing (GD'01)*, volume 2265 of *Lecture Notes in Computer Science*, pages 501–512. Springer-Verlag, 2001. 52

[BEKW02] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. Sketch-driven orthogonal graph drawing. In *Proc. 10th International Symposium on Graph Drawing (GD'02)*, volume 2528 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 2002. 4

[BLR00] Thomas Barkowsky, Longin Jan Latecki, and Kai-Florian Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In C. Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Proc. Spatial Cognition II—Integrating abstract theories, empirical studies, formal models, and practical applications*, volume 1849 of *Lecture Notes in Artificial Intelligence*, pages 41–53, 2000. 14, 15

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997. 11

[BT04] Hans L. Bodlaender and Gerard Tel. A note on rectilinearity and angular resolution. *Journal of Graph Algorithms and Applications*, 8(1):89–94, 2004. 31

[CBD+01] Sergio Cabello, Mark de Berg, Steven van Dijk, Marc van Kreveld, and Tycho Strijk. Schematization of road networks. In *Proc. 17th Annual Symposium on Computational Geometry (SoCG'01)*, pages 33–39, New York, NY, 3–5 June 2001. ACM Press. 2, 14, 15, 48

[CDR03] Sergio Cabello, Erik D. Demaine, and Günter Rote. Planar embeddings of graphs with specified edge lengths. In *Proc. 11th Int. Symp. on Graph Drawing (GD'03)*, 2003. 32

[CR99a] Vijay Chandru and M. R. Rao. *Integer Programming*, chapter 32, pages 32/1–32/45. In Atallah [Ata99], 1999. 11

[CR99b] Vijay Chandru and M. R. Rao. *Linear Programming*, chapter 31, pages 31/1–31/37. In Atallah [Ata99], 1999. 11

[CvK03] Sergio Cabello and Marc van Kreveld. Approximation algorithms for aligning points. In *Proc. 19th Annual Symposium on Computational Geometry (SoCG'03)*, pages 20–28, New York, NY, USA, 2003. ACM Press. 14, 15

[Dan51] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339–247, 1951. 12

[dBETT99] Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999. 4, 7, 10, 16, 29

[EFK01] Markus Eiglsperger, Sándor P. Fekete, and Gunnar W. Klau. Orthogonal graph drawing. In Kaufmann and Wagner [KW01], chapter 6, pages 121–171. 4, 29

[EM99] Peter Eades and Petra Mutzel. *Graph Drawing Algorithms*, chapter 9, pages 9/1–9/26. In Atallah [Ata99], 1999. 7

[Gar94] Ken Garland. *Mr Beck's Underground Map*. Capital Transport Publishing, 1994. 1, 20

[glp] *GNU Linear Programming Kit: Reference Manual Version 4.5*. 53

[gra] The GraphML file format. `http://graphml.graphdrawing.org`. 52

[GT96] Ashim Garg and Roberto Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In *Proc. Int. Symp. Graph Drawing (GD'96)*, volume 1190 of *Lecture Notes in Computer Science*, pages 201–216. Springer-Verlag, 1996. 30

[Hač79] L. G. Hačijan. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 1979. 12

[Had03] Janin Hadlaw. The london underground map: Imagining modern time and space. *Design Issues*, 19(1):25–35, 2003. 1

[HMN05] Seok-Hee Hong, Damian Merrick, and Hugo A. D. do Nascimento. The metro map layout problem. In János Pach, editor, *Proc. 12th Int. Symp. on Graph Drawing (GD'04)*, volume 3383 of *Lecture Notes in Computer Science*, pages 482–491. Springer-Verlag, 2005. 4, 15, 16, 22, 23, 48, 55, 64, 67

[HW02] William C. Hahn and Robert A. Weinberg. A subway map of cancer pathways. `http://www.nature.com/nrc/poster/subpathways/index.html`, 2002. Poster in Nature Reviews Cancer. 4

[ILO] ILOG Inc. *CPLEX 9.0 File Formats Manual*. 53

[IV04] *Proc. 8th International Conference on Information Visualisation (IV'04)*, London, UK, 14–16 July 2004. IEEE Computer Society. 81, 82

[jun] Java universal network/graph framework (JUNG). `http://jung.sourceforge.net`. 52

[Kar] Karlsruher Verkehrsverbund. `http://www.kvv.de`. 62

[Kar84] N.K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. 12

[KM99] Gunnar W. Klau and Petra Mutzel. Combining graph labeling and compaction. In J. Kratochvíl, editor, *Proc. 8th Int. Symp. Graph Drawing (GD'99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 27–37, Štiřín Castle, Czech Republic, September 1999. Springer-Verlag. 16

[KM03] Gunnar W. Klau and Petra Mutzel. Automatic layout and labelling of state diagrams. In Willi Jäger and Hans-Joachim Krebs, editors, *Mathematics – Key Technology for the Future*, pages 584–608. Springer-Verlag, 2003. 16

[KR92] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. on Discrete Mathematics*, 5(3):422–427, 1992. 32

[KW01] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001. 7, 80

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. 31, 32

[ltm] The london tube map archive. `http://www.clarksbury.com/cdl/maps.html`. 25, 26

[Met] Metro de Madrid. `http://www.metromadrid.es`. 27

[Mon96] Mark Monmonier. *Eins zu einer Million*. Birkhäuser, 1996. 20

[Mor96] Alastair Morrison. Public transport maps in western European cities. *The Cartographic Journal*, 33(2):93–110, 1996. 2, 20, 21

[Nes04] Keith V. Nesbitt. Getting to more abstract places using the metro map metaphor. In IV [IV04], pages 488–493. 3

[Ney99] Gabriele Neyer. Line simplification with restricted orientations. In Frank K. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Proc. 6th Int. Workshop on Algorithms and Data Structures (WADS'99)*, volume 1663 of *Lecture Notes in Computer Science*, pages 13–24, Vancouver, BC, 11–14 August 1999. Springer-Verlag. 13

[Nöl05] Martin Nöllenburg. Automated drawings of metro maps. Master's thesis, Institut für Theoretische Informatik, Universität Karlsruhe (TH), August 2005. i

[O'R] O'Reilly. Open source route map. `http://www.oreilly.de/artikel/routemap.html`. 4

[Ove03] Mark Ovenden. *Metro Maps of the World*. Capital Transport Publishing, 2003. 1, 2, 19, 20, 21

[S-B] S-Bahn RheinNeckar. `http://www.s-bahn-rhein-neckar.de`. 72, 73

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986. 11

[Soc] Société de Transport de Montréal. `http://www.stm.info/English/metro/a-mapmet.htm`. 56

[SR04] Jonathan M. Stott and Peter Rodgers. Metro map layout using multicriteria optimization. In IV [IV04], pages 355–362. 4, 15, 22, 23, 48, 55, 64, 66, 67, 68

[SR05] Jonathan M. Stott and Peter Rodgers. Automatic metro map design techniques. In *22nd International Cartographic Conference*, 2005. 16

[SRB+05] Jonathan M. Stott, Peter Rodgers, Remo Aslak Burkhard, Michael Meier, and Matthias Thomas Jelle Smis. Automatic layout of project plans using a metro map metaphor. In *9th International Conference on Information Visualisation (IV05)*. IEEE, 2005. 4

[SW01] Tycho Strijk and Alexander Wolff. Labeling points with circles. *Int. J. Comp. Geom.& Appl.*, 11(2):181–195, 2001. 32

[Syd] Sydney CityRail. `http://www.cityrail.nsw.gov.au/networkmaps/network_map.pdf`. 64

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. 29, 30

[Traa] Transport for London. `http://www.tfl.gov.uk/tube/maps/`. 69

[Trab] Transport for London. London buses spider maps. `http://www.tfl.gov.uk/buses/spiders/borough.asp`. 3

[Tuf90] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990. 21

[Wei97] Robert Weibel. Generalization of spatial data: Principles and selected algorithms. In Marc van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer, editors, *Algorithmic foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, chapter 5, pages 99–152. Springer-Verlag, 1997. 13

[Wie] Wiener Linien. `http://www.wienerlinien.at/WienerStadtWerke/DOWNLOAD/U-NETZ.pdf`. 59

[WS96] Alexander Wolff and Tycho Strijk. A map labeling bibliography. `http://i11www.ira.uka.de/map-labeling/bibliography/`, 1996. 11