Informatik Forsch. Entw. (2007) 22: 23–44
DOI 10.1007/s00450-007-0036-y

REGULÄRE BEITRÄGE

Drawing Subway Maps: A Survey

Alexander Wolff

Received: 10 July 2007 / Accepted: 5 November 2007 / Published online: 29 November 2007

Abstract This paper deals with automating the drawing of subway maps. There are two features of schematic subway maps that make them different from drawings of other networks such as flow charts or organigrams. First, most schematic subway maps use not only horizontal and vertical lines, but also diagonals. This gives more flexibility in the layout process, but it also makes the problem provably hard. Second, a subway map represents a network whose components have geographic locations that are roughly known to the users of such a map. This knowledge must be respected during the search for a clear layout of the network. For the sake of visual clarity the underlying geography may be distorted, but it must not be given up, otherwise map users will be hopelessly confused.

In this paper we first give a rather generally accepted list of rules that should be adhered to by a good subway map. Next we survey three recent methods for drawing subway maps, analyze their performance with respect to the above rules, and compare the resulting maps among each other and to official subway maps drawn by graphic designers. We then focus on one of the methods, which is based on mixed-integer linear programming, a widely-used global optimization technique. This method guarantees to find a drawing that fulfills a subset of the above-mentioned rules (if such a drawing exists) and optimizes a weighted sum of costs that correspond to the remaining rules. The method can draw even large subway networks such as the London Underground in an aesthetically pleasing manner, similar to maps made by professional graphic designers. If station labels are included in the optimization process, so far only medium-size networks can be drawn. Finally we give evidence why drawing good subway maps is difficult (even without labels).

Keywords Graph drawing · Graph labeling · Subway map · Octilinear layout · Mixed-integer program · NP-hard

Mathematics Subject Classification (2000) 05C62 · 90C90 · 68Q17

1 Introduction

A subway map is a schematic drawing of the underlying geographic network that represents the different stations and subway lines of a subway system. Its purpose is to ease navigation in the network for passengers. 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? Where to get off? Exact geography is not only unnecessary for answering these kinds of questions, it can be even hindering. This fact has first been discovered and exploited by Harry Beck, an engineering draftsman, who created the first schematic map of the London Underground in 1933. This map and the fate of Harry Beck are interesting stories in their own right. Garland has devoted a book to them [14], which is very worth reading. Beck designed his map according to a simple set of rules: Meandering 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 less dense

Work supported by grant WO 758/4-2 of the German Research Foundation (DFG).

A. Wolff (✉)
Faculteit Wiskunde en Informatica,
Technische Universiteit Eindhoven,
Postbus 513,
5600 MB Eindhoven, The Netherlands
http://www.win.tue.nl/~awolff

suburbs in order to create more uniform distances between adjacent stations. In spite of all distortion, the network topology and a general sense of the geometry, e.g., a certain relative position between subway stations, is retained. These principles also apply to the majority of contemporary, manually designed subway maps [21, 26].

A transport network can naturally be represented as a graph, where vertices correspond to stations and edges correspond to physical connections between the incident stations. The true location of stations and tracks determines the input layout of the network. This layout is usually planar (otherwise it can be planarized by introducing dummy vertices at junctions) and hence defines a topological input *embedding* by specifying for each vertex the clockwise order of all adjacent vertices. A layout algorithm basically needs to find vertex positions in the plane such that some desired aesthetic criteria are fulfilled or optimized, e.g., the final drawing should preserve the input topology or have few bends along the individual subway lines, and roughly preserve the input geometry. Since subway stations must be labeled, a layout algorithm for the *network* also needs to consider the space that labels require as these labels must neither overlap with each other nor with parts of the network layout.

The notion of a subway map as we discuss it here is an interesting compromise between schematic road maps [8] where vertex positions are (mostly) fixed and "conventional" graph drawing where vertices can go anywhere. The first approach aims at maintaining the user's mental map, the second approach aims at maximizing aesthetics, such as symmetry.

Interestingly enough, the layout principles of subway maps have not only been used in a geographic setting. Sandvad et al. [29] and Nesbitt [22] use the *metro-map metaphor* as a way to visualize abstract information related to the Internet and "trains of thoughts", respectively. The metro-map metaphor has inspired artists and has been used in advertisement, see Figs. 1 and 2, respectively, for particularly nice examples. Stott et al. [32] present a prototype tool to draw project plans in a subway-map style. Technical and engineering applications of schematic graph layouts, which are currently predominated by orthogonal layouts, can also take advantage of octilinear graph drawing. The main benefit of octilinear layouts is that they potentially use less space and fewer bends while still being very tidy. For example in VLSI design the X Architecture [36] is a recent effort for producing octilinear chip layouts. A different application is to compute schematic layouts of *sketches* of graphs, a concept introduced by Brandes et al. [7]. A sketch can be handmade or the physical embedding of a geometric network like the real position of telephone cables. Brandes et al. give an efficient algorithm for computing an *orthogonal* drawing of a sketch. However, their algorithm cannot be extended to more than four directions. This is another possible application area for methods that can draw subway-style maps.

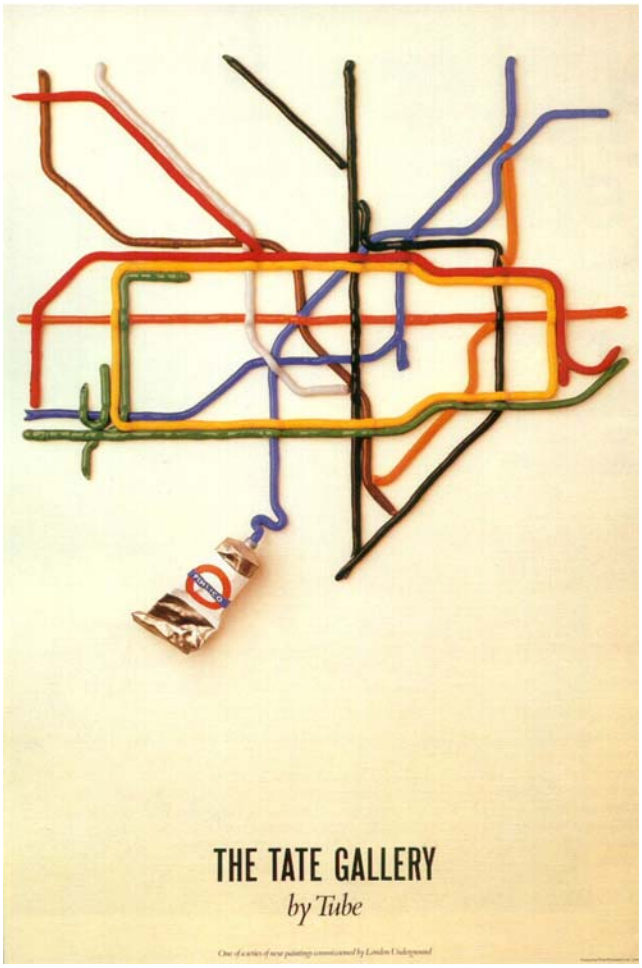


Fig. 1 Poster for the Tate Gallery

Overview. This article is structured as follows.

First, we give a rather generally accepted list of rules that should be adhered to by a good subway map, see Sect. 2.

The main contribution of this article is a survey of three methods for drawing subway maps that have recently been suggested. All of them rely on some underlying optimization machinery, which is tuned in order to get drawings that fulfill the above rules as best as possible. The first of the three methods, by Hong et al. [15], is based on a *spring embedder*, a force-directed graph-layout method. The attracting and repelling forces that drive the movement of the vertices stem from a physical model. They are computed incrementally by a simulated-annealing-like local-optimization algorithm. The second method, by Stott and Rodgers [30], uses multi-criteria optimization based on hill climbing, a popular general-purpose local-optimization technique. The third method, by Nöllenburg and Wolff [25], relies on mixed-integer programming, a widely used global-

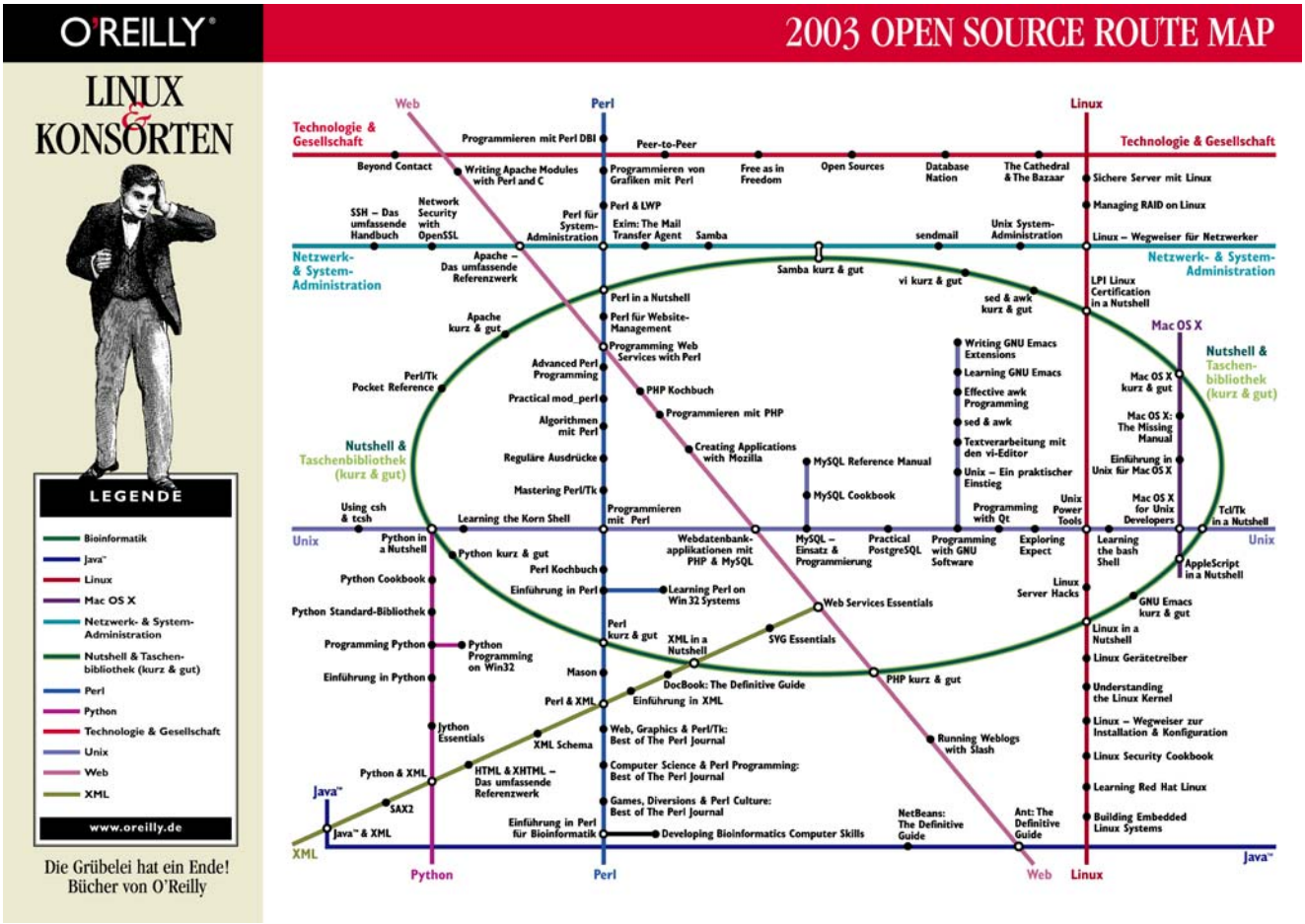


Fig. 2 Open source product lines of the publisher O'Reilly. Interchange stations represent books that simultaneously belong to two product lines

optimization technique. Mixed-integer programming is very powerful, but care needs to be taken to avoid long running times. We analyze the performance of the three methods with respect to the above list of rules and compare their output at a benchmark (the CityRail network of Sydney) that has been tested by all of them.

Next we focus on the method based on mixed-integer programming [25] since it is the only method that guarantees octilinearity, which, in our opinion, is essential for a clear layout of subway maps, see Sect. 4. It is also the first method dedicated to drawing subway maps that uses global optimization and thus avoids getting trapped in local minima. This contrasts with the other two methods based on local optimization. However, since no fast algorithm for solving general mixed-integer programs (MIPs) is known, we sketch some heuristic data reduction and speed-up methods, which are important for solving larger instances. We also sketch how to extend the basic MIP to combine graph drawing with the placement of non-overlapping station labels.

This combined discipline has been called *graph labeling* by Klau and Mutzel [17]. Klau and Mutzel [17] and Binucci et al. [6] have used MIP formulations for graph labeling before. However, their methods follow Tamassia's *topology-shape-metrics approach* [34], which is a common approach for drawing graphs orthogonally. The approach consists of three steps: planarization, orthogonalization, and compaction. The first step fixes the embedding, the second step its shape (for each edge the sequence of its bends and their angles is determined), and the third step the coordinates of the vertices and the bends. Tamassia [34] mentions that his approach for orthogonal graph drawing carries over to hexagonal (i.e., 60°-) drawings, but that the third step fails for drawings with smaller angles (such as 45° in the case of octilinear drawings).

As a justification for the use of the heavy-weight MIP machinery, we then give Nöllenburg's beautiful proof [24] of the NP-hardness of a restricted version of the subway-layout problem, namely deciding whether a given embedded graph can be drawn using straight octilinear edges. The proof reminds of the mechanical constructions that boys used to build with a Meccano or Märklin model construction kit, see Sect. 5. The hardness of octilinear graph drawing is in sharp contrast to orthogonal graph drawing,

which Tamassia [34] showed to be efficiently solvable by his topology-shape-metrics approach.

We conclude with some thoughts about the remaining differences between hand- and machine-made subway maps, and give an open problem in Sect. 6.

Before turning to our list of rules for good subway maps, we refer the interested reader to a very nicely written general-audience (German) newspaper article [27] about the drawing of subway networks. It includes some interesting historic notes.

2 Rules

In this section we list and motivate rules that a good subway map should adhere to. Each of the rules is either implicit or explicit in at least two of the papers of Hong et al. [15], Stott and Rodgers [30], and Nöllenburg and Wolff [25], whose algorithms we will discuss in the next section. The reader is invited to study the book of Ovenden [26], which contains an abundance of subway maps from all over the world, in order to make up his own mind as to which set of rules is the right one. It is an interesting cartographic question whether the rules behind existing subway maps are in fact the most user-friendly choices. For example it is hard to estimate distances or travel times in a subway map. Pairs of stations with the same distance in the subway map might actually be several kilometers apart in peripheral parts but only a few hundred meters in the city center. User studies ought to be made in order to evaluate to which extend current subway maps actually support subway passengers in quickly making the right decisions.

Before listing the rules, we quickly fix some notation roughly following Di Battista et al. [10]. Given a graph $G = (V, E)$ we say that δ is a *drawing* of G if δ maps each vertex v of G to a distinct point $\delta(v)$ of the plane and each edge $\{u, v\}$ to a simple (Jordan) curve that connects $\delta(u)$ and $\delta(v)$. A drawing is *plane* if for any pair of edges, the corresponding curves have at most endpoints in common. Recall that a graph is *planar* if it has a plane drawing. A plane drawing partitions the plane into connected regions called *faces*. The unbounded face is also referred to as *outer* face. An *embedding* is a useful abstraction of plane drawings: it fixes the circular ordering of the edges around each vertex and the choice of the outer face. Now we can say that a graph is *plane* if is planar and is given with a (plane) embedding.

We assume that we are given a simple plane graph G, to which we refer as the *subway graph*. We also assume that we are given a location $\pi(v) \in \mathbb{R}^2$ for each vertex v of G. These locations will usually be the locations of the subway stations on a geographic map. (Note that the locations do *not* define the embedding since we do not assume that the subway graph has straight-line edges.)

(R1) Keep the input embedding. This supports the mental (network) map of the passengers.

(R2) Restrict all line segments to the four *octilinear orientations* horizontal, vertical, and both diagonals at 45°. Each orientation has two *directions*. This restriction makes maps clearer.

(R3) Ensure that adjacent and non-adjacent stations keep a certain minimum distance. This increases the readability of the map.

(R4) Keep the number of bends along a given subway line small, especially in interchange stations where several lines meet. If bends cannot be avoided, obtuse angles are preferred over acute angles, i.e., the order of preference is 135°, 90°, and 45°. This rule helps passengers to follow a subway line with their eyes.

(R5) Preserve the *relative position* between subway stations. For example, a station being north of some other station in reality should not appear below that station on the map. This supports the (geographic) mental map of the passengers.

(R6) Keep the total edge length of the network small. This indirectly makes sure that dense regions of the map get a larger share of the available space. Together with rule (R3) this also keeps distances between adjacent stations as uniform as possible. Rule (R6) supports the clarity of the layout.

(R7) Color each edge according to the lines to which it belongs. This assumes that each line has a unique color. If an edge $\{u, v\}$ belongs to k lines, then k copies of that edge (so-called *multi-edges*) must be drawn. Their order along $\{u, v\}$ should be as consistent as possible with orders along other edges incident to u or v. Coloring is essential to help map users to follow a line with their eyes.

(R8) Label stations with their names, and make sure that labels do not obscure other labels or parts of the network. Preferably all labels between two interchange stations are placed on the same side of the line; stations on a horizontal line may also be alternatingly labeled above and below the line to save space. Labels are essential for a readable map.

Clearly, each subway map can only be a compromise of the above criteria. For example, a map with a minimum number of line bends could drastically distort the mental map and, conversely, preserving the mental map could require a large number of bends.

Now we want to state the *subway-map layout problem* as formally as possible at this point. Let $\mathcal{L}$ be a *line cover* of G, i.e., a set of paths and cycles of G such that each edge of G belongs to at least one element of $\mathcal{L}$. An element $L \in \mathcal{L}$ is called a *line* and corresponds to a subway line of the underlying transport network.

To keep the problem description concise, we do not insist on coloring (multi-) edges (rule (R7)) and placing station labels (rule (R8)) for now. Subway lines still play a role when it comes to counting bends, see rule (R4). The ordering of the (line-colored) multi-edges along an edge of G is an interesting research topic by itself and has found some attention recently [3, 4]. For more on label placement, see Sect. 4.6.

Problem 1 (Subway-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, find a *nice* drawing of G, i.e., a straight-line drawing that follows rules (R1)–(R6) as much as possible.

Note that the restriction to graphs with maximum vertex degree 8 is an immediate consequence of the restriction to octilinear edge directions. Lifting this restriction is discussed in the conclusions (Sect. 6).

3 Methods

There are three recent approaches to automating the drawing of subway maps. We survey these methods in order of date of first publication. All of them rely on some underlying optimization machinery, which is tuned in order to get drawings that fulfill the above rules as best as possible. The method of Hong et al. [15] is based on a spring embedder, the method of Stott and Rodgers [30] uses hill climbing, and the method of Nöllenburg and Wolff [25] relies on mixed-integer programming.

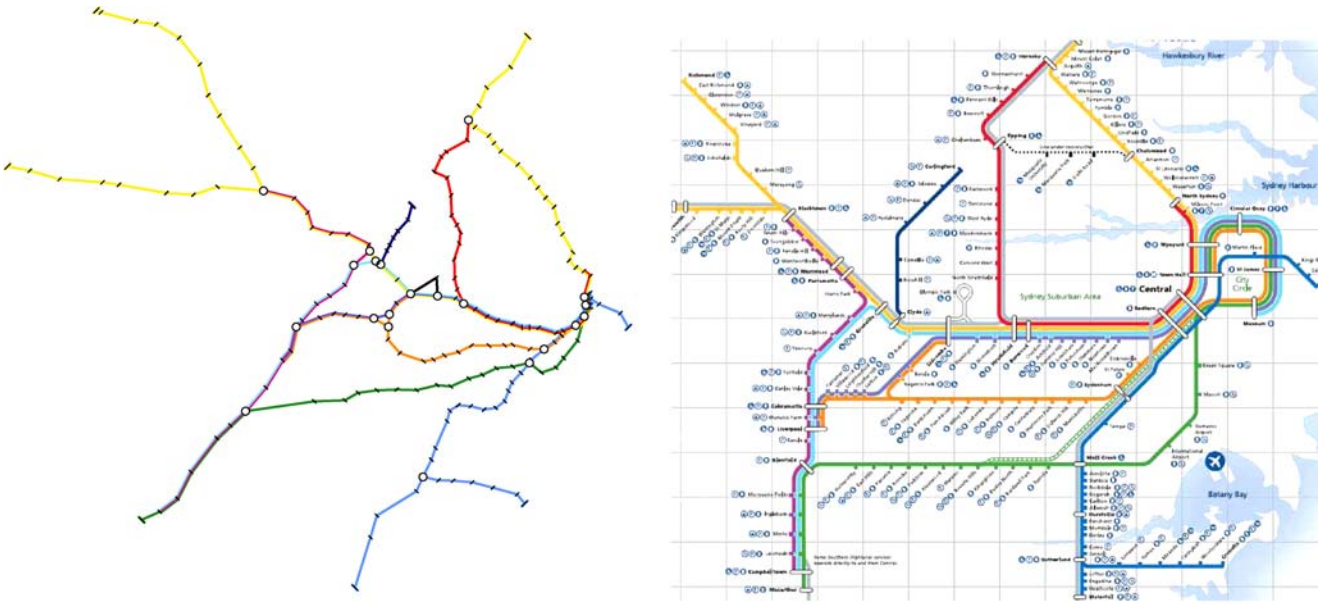
As benchmark we use the urban part of the CityRail network of Sydney, a medium-size transportation network. The reason for this choice is that the Sydney subway graph is the only that appears in all three articles. The fact that our comparison is based on a single network seems to be rather restrictive. However, when we compared for a given method its drawing of our benchmark network to its drawings of other networks, we found that the typical features of the method indeed show up in the benchmark drawing.

Figure 3a shows the geographic position of the stations (with straight-line edges connecting them); Figure 3b shows the corresponding clipping of the – octilinear! – official map made by graphic designers. It is instructive to compare edge lengths in the downtown area (around the station Central) with edge lengths in the suburbs on each of the two maps. Table 1 describes the combinatorial size of the network before and after a preprocessing step that is detailed below.

For each method we detail the computing environment used and the running times reported in the corresponding article. Although the set-ups were of course not identical, all methods have been implemented in Java and were run on single-processor machines with clock rates of 2.2–3 GHz and 0.5–4 GB RAM. Thus it is likely that a running time in the order of seconds or minutes on one machine would have

Table 1 Numbers n, m, m', and f of vertices, edges, multi-edges, and faces of the Sydney subway graph, respectively

G	n	m	m'	f
original	174	183	289	11
contracted [15]	31	40	40	11
contracted [25]	67	76	145	11



a Geographic layout **b** Corresponding clipping of the official map [34]

Fig. 3 The Sydney CityRail subway network

remained of the same order on another. One can argue that running time is not critical when drawing subway maps that usually have a life expectancy of several years. However, if the new methods for octilinear drawing are to challenge conventional methods for orthogonal drawing (of class diagrams in software projects, for example), then speed may become more important than aesthetics. Certain use cases, such as on-line mapping, are ruled out by methods that usually need more than a few seconds to produce a result.

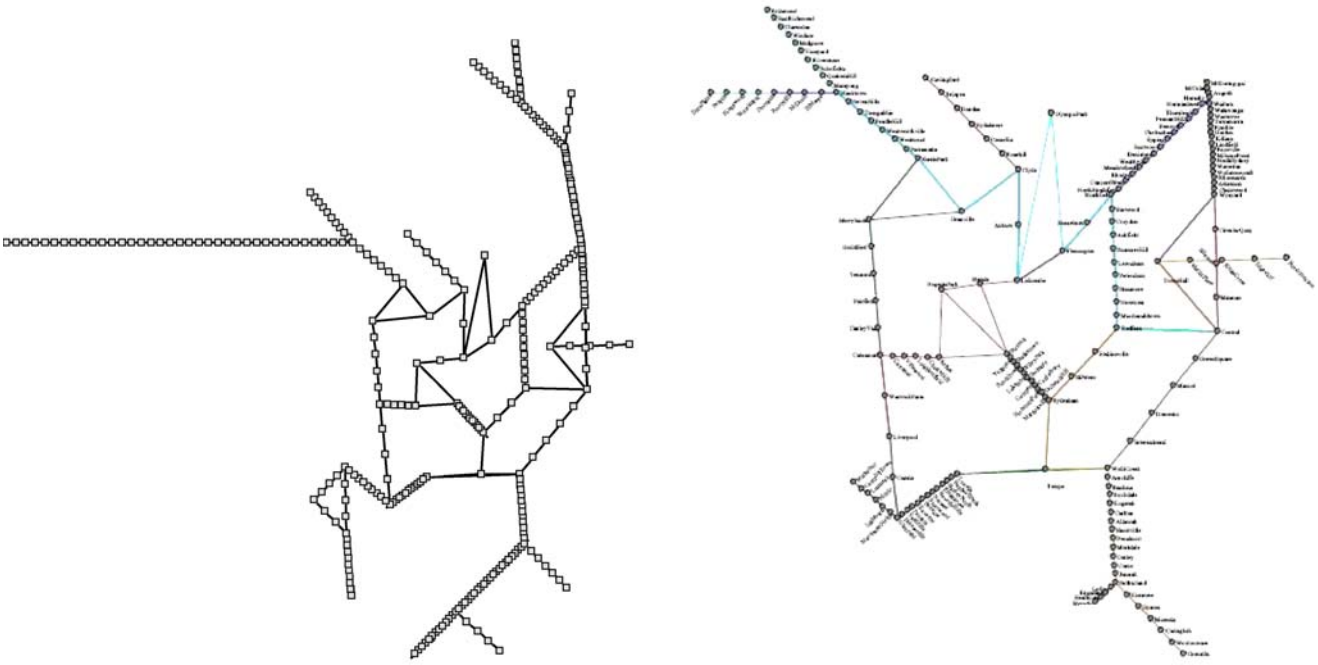
3.1 Spring embedder

The first method, by Hong et al. [15], is based on a spring embedder. A spring embedder is an iterative algorithm that simulates a physical system that in turn represents the graph to be drawn. One can think of the vertices as particles with repelling forces and of the edges as springs with contracting forces. This very popular and fast all-round graph-drawing algorithm was suggested by Fruchterman and Reingold [12] in the early 1990's.

There are plenty of spring-embedder variants available. One of them is PrEd [5], which Hong et al. chose as basis of their algorithm since PrEd maintains the embedding of the input graph. Hong et al. give five algorithms to address the subway-map layout problem. The most refined of these algorithms modifies PrEd such that edge weights are taken into account and such that additional magnetic forces draw the straight-line edges towards the closest octilinear direction. As in the original version of PrEd, the total force acting on a vertex is simply the sum of all attracting and repelling forces that act on that vertex. Some of the expressions for the partial forces include model-tuning parameters; the authors mention seven for the x-components of the partial forces alone.

Hong et al. consider the geometry of the input network only implicitly: they use the original embedding as initial layout. In a preprocessing step they simplify the subway graph by collapsing all degree-2 vertices. See Table 1 for the effect of this. Then they set the weight of each edge e in the remaining graph to the number of original edges that e replaces. Having computed the final layout, Hong et al. reinsert all degree-2 vertices into the corresponding edges in an equidistant manner. Their algorithm is very fast: all their examples were computed within a few seconds on a single-processor 3.0 GHz Pentium-4 machine with 1 GB of RAM under the Sun Microsystems Java-2 Runtime Environment, Standard Edition. The Sydney CityRail network with fixed embedding took 7.6 s. Station labels are placed in a second, independent step. While label–label overlaps are avoided, labels sometimes do intersect edges.

Figure 4b is a clipping of Fig. 17 in the article of Hong et al. [15] and shows their Sydney CityRail map. Originally they draw a slightly larger labeled network (for the unlabeled counterpart, see Fig. 4a), where lines extend further into the suburbs, but we only consider the part corresponding to the graph in Fig. 3a. If the depicted drawing is to be



a Complete unlabeled map

b Clipped labeled map [16]. The clipping corresponds to the maps in Fig. 3

Fig. 4 Drawings of the Sydney CityRail network by Hong et al.

judged with respect to the rules (R1)–(R6), it is clear that the drawing preserves the embedding, i.e., (R1) is fulfilled by construction. It further seems that relative position (rule (R5)) is respected quite well. This is probably due to the fact that the geographic layout was the starting point of the iterative layout process. On the other hand one can observe that edges are not strictly octilinear (R2) and that there is a large variance in the distribution of the edge lengths. If the smallest edge length is assumed to be unit (R3), then the total length of the network is huge (R6). This may be due to the fact that the above mentioned forces that determine the layout are sums of many conflicting terms. Most lines bend at most interchange stations more or less abruptly (rule (R4)); see the (light blue) line that goes from the topmost station Richmond to Central (right middle), for example. This does not come as a surprise: line bends are not taken into account by the magnetic forces that Hong et al. define.

3.2 Hill climbing

The second method, by Stott and Rodgers [30], uses multicriteria optimization based on hill climbing, a popular general-purpose local-optimization technique. They first map the geographic layout of the given subway network to an integer grid. To evaluate a drawing they use a metric which is the weighted sum of five sub-metrics each of which measures a specific aesthetic value of a drawing.

Stott and Rodgers [30] draw subway maps using multicriteria optimization based on hill climbing. For a given layout they define metrics that evaluate the number of edge intersections, the octilinearity, the edge lengths, the angular resolution at vertices, and the straightness of subway lines. Then they define the quality of a layout to be the weighted sum over these five metrics. Their iterative optimization process starts with a layout on the integer grid that is obtained from the original embedding. In each iteration they consider alternative grid positions for each vertex and 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 and preserves the topology, they move the current vertex to the position with the largest improvement.

One can view this approach also as follows. Call two drawings *neighbors* if they only differ in the position of one vertex and if the two positions lie on the same grid square. The quality measure mentioned above defines a landscape over the graph of neighboring drawings; here hill climbing yields (at least local) extrema.

Stott and Rodgers observe typical problems with local extrema during their optimization process and give a heuristic fix that overcomes one of these problems: they shorten each overlong bridge b by moving the smaller component of $G - b$ by an appropriate amount towards the larger. Stott and Rodgers optionally use a similar edge contraction step as Hong et al. [15] to preprocess the input graph. Even with this preprocessing their algorithm is much slower than that of Hong et al.; the Sydney graph took them roughly 24 min without and 4 min with preprocessing on a 2.4 GHz Pentium-4 machine with 512 MB RAM under Java 2 v1.4.2.

For the results, see the maps in Fig. 5. These maps fulfill most of our rules quite well. Rule (R4) is one of the exceptions; especially in Fig. 5a there are many unnecessary bends, but also in Fig. 5b most lines bend in most interchange stations. Again, take the (here yellow) line from Richmond to Central as an example. Recall that one of the five metrics that Stott and Rodgers use to define the quality of a layout in fact punishes the number of bends. Increasing the weight of this metric would probably yield a map with fewer bends – maybe at the expense of the other metrics.

There seems to be an interesting trade-off between rules (R2) and (R6): the map that was drawn without the edge-contraction step (Fig. 5a) contains only one non-octilinear edge, but a rather high variance of edge lengths, while the other map (Fig. 5b) has three non-octilinear paths of degree-2 vertices, but more or less uniform edge lengths (due to the edge-contraction step).

The first of their five metrics punishes intersections, which means that a non-plane drawing can become plane during the layout process. This is what happened to the intersection between the city circle and the blue line – the last three stations which actually lie outside the circle (see the rightmost stations in Fig. 3b versus those in Fig. 5) are moved inside. The idea behind this metric is that it should remove intersections that may sometimes come into being since the initial layout uses geographic station positions and straight-line edges. However, in the Sydney example it would probably have made sense to do the obvious: insert a dummy vertex at the intersection and call the layout algorithm (possibly punishing bends at dummy vertices harder). This approach may result in unwanted bends (at points other than stations), but at least the network topology would be correct.

Stott and Rodgers [31] extend their previous method by integrating station labeling into their optimization process. After each iteration of vertex movements the number of intersections between labels and edges, vertices, and other labels is minimized. In spite of this, they experience quite a few label–label overlaps, especially along horizontal edges. The authors do not give information on the running time of their method for labeled subway maps. The networks they draw and label are rather small (at most five lines) and, more importantly, have few faces (one to three, including the outer face).

3.3 Mixed-integer programming

The third method, by Nöllenburg and Wolff [25], is based on mixed-integer linear programming, a widely used global-

optimization technique. A mixed-integer linear program (MIP) consists of (a) integer and real variables, (b) a linear objective function, i.e., a weighted sum of the variables, and (c) linear constraints, i.e., equalities or inequalities that have a weighted sum of the variables on one side and a constant on the other. In contrast to linear programming, where integer variables are not allowed, mixed-integer programming is very versatile and can be used to model many problems. Since this includes NP-hard problems, no polynomial-time algorithm for general MIPs is known. However, there is a number of public-domain (e.g., lp_solve) and commercial solvers available (e.g., CPLEX). The size of the instances that these programs can solve and the resulting running times depend heavily on the problem. However, even if it may take a long time to solve a MIP to optimality, a MIP solver usually finds better and better *feasible* solutions in the process, i.e., solutions that fulfill all (integrality and other) constraints, but which may not be optimal. This is of interest for practical problems such as the drawing of subway maps, where optimality is often not required. Moreover, with each feasible solution MIP solvers output a so-called *optimality gap*, i.e., the relative gap between the cost of the feasible solution and the best lower bound currently proven by the MIP solver. This is a valuable quality estimate.

Nöllenburg and Wolff [25] model the problem by splitting the list of rules. They refer to rules (R1)–(R3) as *hard* constraints and to rules (R4)–(R6) as *soft* constraints. The hard constraints are those that Nöllenburg and Wolff insisted on, while the soft constraints model aesthetic criteria that are to be optimized among all drawings that fulfill the hard constraints. Given these two categories, the hard constraints were translated into linear equalities and inequalities of a set of variables (basically the coordinates of the vertices). Then the soft constraints were translated into a weighted sum of three cost functions that measure how well these constraints are fulfilled. These are the ingredients of a MIP formulation: variables, linear constraints, and a linear objective function. This translation of rules into a formulation that only allows *linear* expressions is probably the most difficult part of the work, see Sect. 4.

Given the MIP formulation and a concrete input – the embedding of the input graph, the locations of the vertices, and the line cover, – the corresponding MIP instance can be generated automatically. Then a MIP solver is called. If the MIP instance is not too large, the problem not too difficult, and the solution space not empty, the MIP solver will find feasible solutions of better and better quality and eventually the optimal solution. Note that here a feasible solution corresponds to a drawing that meets all hard constraints, and an optimal solution corresponds to a drawing that additionally optimizes the soft constraints.

The MIP approach yielded the planar layout in Fig. 6a. Nöllenburg and Wolff [25] report a computation time of 22 min that was based on a simple ad-hoc heuristic for reducing the size of the MIP. In the meantime – given some more engineering (see Sect. 4.5) – the computation time has

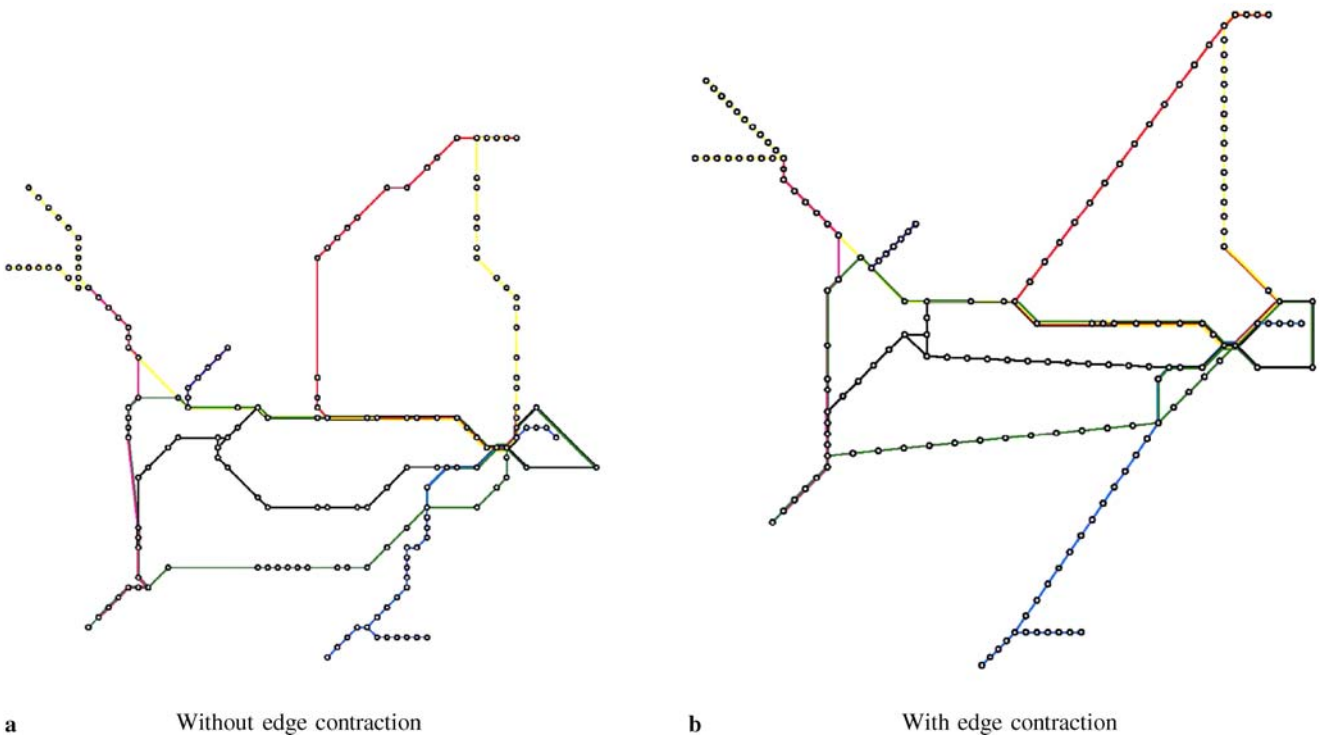


Fig. 5 Drawings of the Sydney CityRail network by Stott and Rodgers [30]

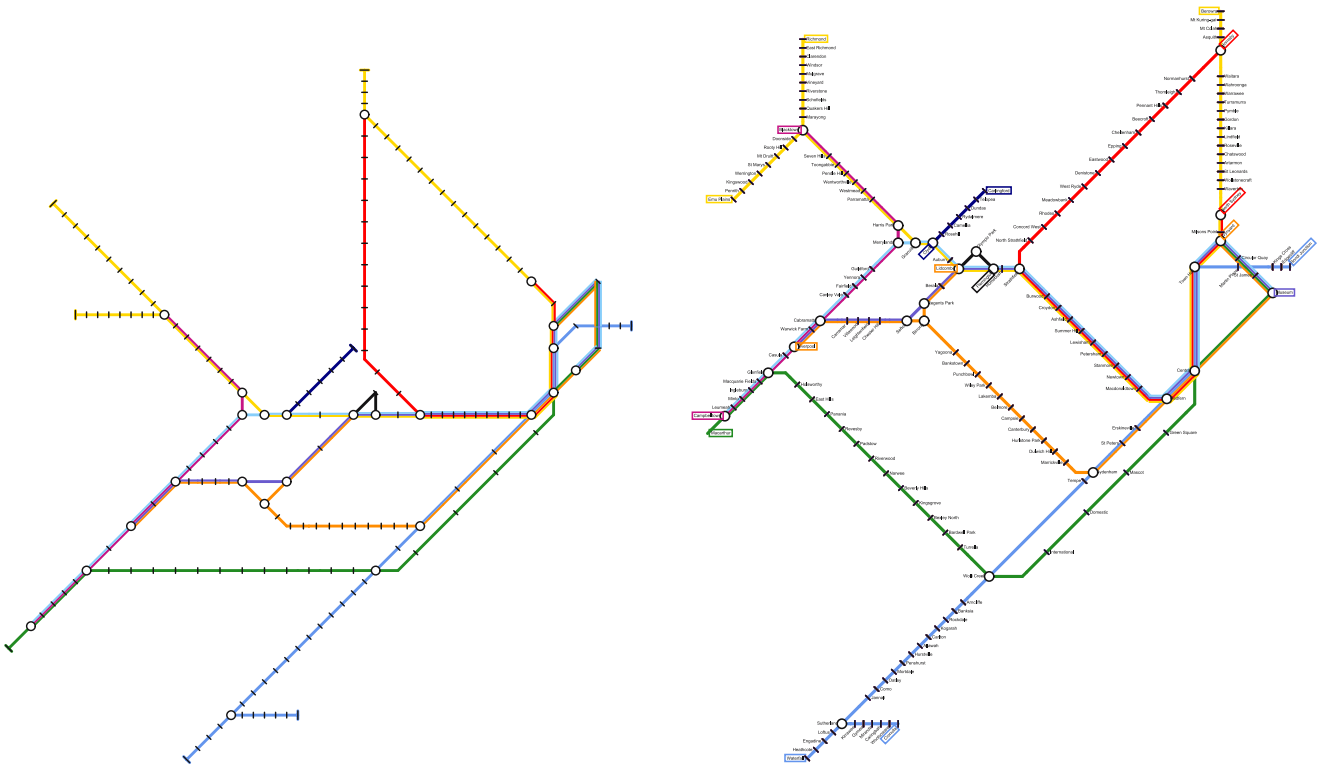
been reduced to 77 s with the MIP solver CPLEX 9.1 running on an Opteron-248 processor with 2.2 GHz and 4 GB RAM under SuSE Linux 9.3. Optimality in terms of the objective function was not proven. For this solution the optimality gap was 26.4%. In other words, we know that the optimal drawing (which we do *not* know) achieves a value of the objective function that is at most 26.4% less than that of the drawing in Fig. 6a. (Recall that over time, the MIP solver does not only produce better and better feasible solutions, but also tighter and tighter bounds on the optimum.) Note, however, that the error in modeling human perception by the *choice* of the objective function (1) is probably much larger.

By construction the drawing in Fig. 6a fulfills the hard constraints, i.e., rules (R1)–(R3). Observe the influence of the soft constraints on the layout: there are no unnecessarily long edges (rule (R6)); the subway lines only bend where geographically required and pass through interchange stations as straight as possible (rule (R4)); and, finally, the simplified edges tend to follow the original directions (rule (R5)).

Computing a labeled subway map of Sydney took much longer; the result after 4 h and 57 min with an optimality gap of 32.3% is shown in Fig. 6b. By construction of the MIP there is no overlap between labels and any other object in the drawing. Note that many of the horizontal edges in Fig. 6a are drawn diagonally in the labeled layout. For more on labeling, refer to Sect. 4.6.

3.4 Related map-schematization methods

Apart from the above three papers that tackle subway maps explicitly, there are some related papers on map schematization. Neyer [23] studied a line simplification problem for polygonal paths and gave a polynomial-time algorithm to find approximations to these paths using only a restricted number of orientations. Barkowsky et al. [2] used *discrete curve evolution*, an algorithm for polygonal line simplification, to draw schematic maps. As one example they looked at the lines of the Hamburg subway system. However, their algorithm neither restricts the edge directions nor does it increase station distances in dense downtown areas. Stations are labeled but no effort is made to avoid label overlap. Avelar and Müller [1] implemented a simulated-annealing algorithm to modify a given input map by iteratively moving the endpoints of line segments such that edges approach octilinear line segments. The algorithm was 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. Ware et al. [35] built on the work of Avelar and Müller and tailor



a Unlabeled map **b** Labeled map

Fig. 6 Drawings of the Sydney CityRail network by Nöllenburg and Wolff

it towards mobile applications with small screens. Cabello et al. [8] presented an efficient algorithm for schematizing road networks. Their algorithm draws edges as octilinear paths with two or three links and preserves the input topology. However, all vertices keep their original positions, which is in general not desired for drawing subway maps. Cabello and van Kreveld [9] studied approximation algorithms for aligning points octilinearly, where each point can be placed anywhere in its own given region. Yet, their method does not guarantee to preserve the embedding if points correspond to vertices of a graph. Merrick and Gudmundsson [20] suggested an efficient algorithm that simplifies polygonal chains using the following criteria. Given a polygonal chain P, a fixed set C of directions, and an accuracy ε, their algorithm finds a polygonal chain P' that is (a) C-oriented, (b) goes through all radius-ε centered at vertices of P in the right order, and (c) has the minimum number of bends among all chains that fulfill (a) and (b). They apply their algorithm to drawing subway networks. However, since the algorithm processes each subway line individually, the input topology is not kept.

4 Mixed-integer program

Nöllenburg and Wolff formulate the subway-map layout problem as a MIP. As we will see in Sect. 5, the subway-map layout problem is NP-hard. This is a good justification for applying a likewise NP-hard optimization method such as mixed-integer programming. The expressive power of mixed-integer programming gives 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 that Nöllenburg and Wolff tried), then their MIP finds such a layout. At the same time, their MIP optimizes the weighted sum of cost functions each of which corresponds to a soft constraint. Before we describe a number of features of this MIP we give some basics on linear programming.

4.1 Basics

A MIP consists of two parts: a set of linear constraints and a linear objective function. We give a simple two-dimensional example, see Fig. 7.

Consider the objective function

$$\text{maximize } x + 2y \tag{1}$$

subject to the constraints

$$y \le 0.9 \cdot x + 1.5 \tag{2}$$

$$y \ge 1.4 \cdot x - 1.3\,. \tag{3}$$

Each constraint corresponds to a halfplane; the intersection of the halfplanes – a (possibly unbounded) polygon – represents the set S of feasible solutions. In Fig. 7 the solution polygon is the shaded region. Among the points in S we are interested in one that maximizes the objective function, which also has a geometric interpretation. The coefficients of the variables in the objective function yield a vector c, here $c = \binom{1}{2}$. If we sweep the plane in direction c with a line ℓ orthogonal to c, then the last points of S swept by ℓ are those that maximize (1). The traces of ℓ are marked by the dashed lines in Fig. 7.

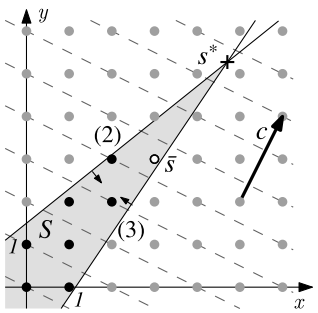


Fig. 7 Difference between optimal fractional solution s^* and optimal integral solution $\bar{s}$

Objective function (1) and constraints (2) and (3) represent in fact a *linear program (LP)*. LPs can be solved efficiently, e.g., by Karmakar's interior-point method [16]. This changes radically when we add *integrality constraints*, e.g.,

$$x, y \in \mathbb{Z}. \tag{4}$$

Then we get an *integer (linear) program (IP)*, whose solution set consists of those points in S that lie on the integer grid. In our example in Fig. 7 these points are marked by black dots. Note that the optimum solution $\bar{s}$ of the IP (1)–(4) is usually far from the optimum solution s^* of the LP (1)–(3); the optimum integral solution $\bar{s}$ can *not* be obtained from the optimum fractional solution s^* by rounding down the components of the vector s^*. A MIP can have both, fractional and integral variables.

Integrality constraints make a continuous problem discrete; if the set of fractional solutions is bounded, then the number of integral solutions becomes finite. So it seems solving the more restricted problem is easier. However, the opposite is the case. Geometric properties of the LP that are exploited by efficient solution strategies are lost. On the other hand a lot more problems can be modeled with the help of integer variables.

We give an example that will come in handy later on, a standard trick in MIP modeling. Suppose we want to make sure that at least one of three constraints C_1, C_2, and C_3 is fulfilled, but not necessarily all of them. In other words, we want to express the disjunction $C_1 \vee C_2 \vee C_3$. Suppose

$$C_1: \quad x - 3 \le 0\,,$$
$$C_2: \quad y \le 0\,,$$
$$C_3: \quad x + y \le 0\,.$$

Then we introduce three binary variables α_1, α_2, and α_3, i.e., variables that are restricted to the set $\{0, 1\}$. We further restrict these variables by the constraint

$$\alpha_1 + \alpha_2 + \alpha_3 \geq 1\,. \tag{5}$$

Now we can formulate $C_1 \vee C_2 \vee C_3$ as the *conjunction* $C_1' \wedge C_2' \wedge C_3'$, where

$$\begin{aligned} C_1' &: \quad x - 3 \leq M(1-\alpha_1)\,, \\ C_2' &: \qquad y \leq M(1-\alpha_2)\,, \\ C_3' &: \quad x + y \leq M(1-\alpha_3) \end{aligned} \tag{6}$$

and M is a large constant that must be an upper bound on the left-hand sides of the inequalities. Note that (5) and (6) are a conjunction of linear constraints, i.e., legal part of a MIP. By the way: it is worth making M as tight a bound on the left-hand sides as possible – this helps to speed up solving the MIP.

4.2 Overview

In the remainder of this section we peek into the MIP formulation of Nöllenburg and Wolff [25]. We have chosen two parts. We first detail how the rather subway-specific hard constraint *octilinearity* (rule (R2)) is modeled, see Sect. 4.3. Then, we turn to the soft constraint *relative position* (rule (R5)), which is also specific to drawing geometric networks such as subway maps, see Sect. 4.4.

Recall that Nöllenburg and Wolff (roughly) translate the three hard constraints (R1)–(R3) into the linear constraints of their MIP, and the three soft constraints (R4)–(R6) into the objective function. The objective function is simply the weighted sum of three individual cost functions:

$$\text{Minimize } \lambda_{R4}\,\text{cost}_{R4} + \lambda_{R5}\,\text{cost}_{R5} + \lambda_{R6}\,\text{cost}_{R6}\,, \tag{7}$$

where the constants λ_{R4}, λ_{R5}, and λ_{R6} can be set by the user. Each of them individually emphasizes a certain aesthetic criterion. The function cost_{R5} responsible for optimizing relative position is treated in detail in Sect. 4.4, for the other two cost functions see [25].

The total number of constraints and variables in the MIP formulation of Nöllenburg and Wolff is $O(n + m' + m^2)$, where $m' > m$ is the sum of the number of edges in all lines in $\mathcal{L}$ (counting multiple edges). Note that since G is planar, Euler's polyhedral formula yields $m \leq 3n - 6$. Section 4.5 deals with methods for reducing the size of the subway graph and of the MIP. Section 4.6 describes how vertex labels can be included in the MIP model. Section 4.7 describes how the speed-up techniques and the integration of label placement affect running time and results.

We close this overview by presenting the coordinate system that we are going to use. The idea is that we would like to handle all four edge orientations similarly. Hence we use an (x, y, z_1, z_2)-coordinate system as depicted in Fig. 8, where each axis corresponds to one of the four edge orientations in the layout. For each vertex $v \in V$ we set

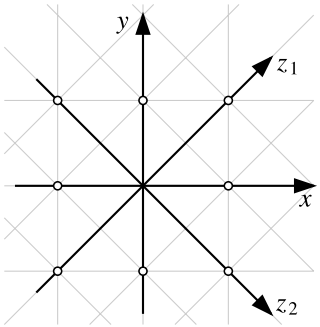


Fig. 8 The octilinear coordinate system with an octilinear grid in the background. The marked points all have the same L_∞-distance from the origin

$$\begin{aligned} z_1(v) &= x(v) + y(v) \\ z_2(v) &= x(v) - y(v)\,. \end{aligned} \tag{8}$$

These defining equations are also part of the MIP formulation and use real-valued variables for the coordinates.

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 Fig. 8 eight points on the octilinear coordinate axes are shown that all have the same L_∞-distance from the origin. One side-effect of using the L_∞-metric is that all vertices will be placed on a axis-aligned 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 an L_∞-distance of 1.

4.3 Enforcing octilinearity and relative position

The following part of the MIP formulation models hard constraints, namely that all edges are drawn as straight, octilinear line segments with a given minimum length, as stated in rules (R2) and (R3). Note that rule (R3) is only partially covered here. The distance requirement for non-adjacent vertices is more naturally handled together with the enforcement of planarity, see [25].

At the same time, the direction of each edge $\{u, v\}$ in the output drawing is restricted to the three closest octilinear approximations[1] of the line segment $\overline{\pi(u)\pi(v)}$ specified by the input. Hence, the soft constraint (R5) is partially modeled as a hard constraint, too. Relative position can also be modeled completely as a soft constraint, but excluding

[1] This means that the angle between the directed lines through u and v in in- and output is at most 67.5°.

a number of directions has the potential of speeding up the solving of the MIP.

Before formulating the constraints we need some notation to address relative positions between vertices and to denote directions of edges. For technical reasons we direct all edges arbitrarily. If we write an edge as uv, we mean that it is directed from u to v. For each vertex u we define a partition of the plane into eight sectors. Each sector is a 45°-wedge with apex u. The wedges are centered around rays that emanate from u 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, see Fig. 9.

To denote the rough relative position between two vertices u and v in the *original layout* we use the terms $\mathrm{sec}_u(v)$ and $\mathrm{sec}_v(u)$ representing the sector relative to u in which v lies and vice versa. Note that these terms are known before solving a concrete instance and are thus *constants* from the MIP point of view. For each edge uv, we introduce the *variable* $\mathrm{dir}(u, v)$ to denote the octilinear direction of uv 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 $\mathrm{dir}(u, v) = 5$. To make the difference between $\mathrm{sec}_u(v)$ and $\mathrm{dir}(u, v)$ really clear, note that the former describes the (known) input, while the latter describes the (unknown) output. Both fulfill a kind of symmetry:

$$\mathrm{sec}_u(v) = \mathrm{sec}_v(u) + 4 \pmod 8 \quad \text{and}$$
$$\mathrm{dir}(u, v) = \mathrm{dir}(v, u) + 4 \pmod 8 .$$

As mentioned above, we partially model the soft constraint (R5) as a hard constraint. As a compromise between conservation of relative positions and flexibility to obtain a nice drawing, we allow that an edge is drawn in one of 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. Let

$$\mathrm{sec}_u^{\mathrm{pred}}(v) = \mathrm{sec}_u(v) - 1 \pmod 8 ,$$
$$\mathrm{sec}_u^{\mathrm{orig}}(v) = \mathrm{sec}_u(v) ,$$
$$\mathrm{sec}_u^{\mathrm{succ}}(v) = \mathrm{sec}_u(v) + 1 \pmod 8 .$$

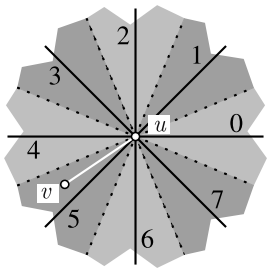


Fig. 9 Numbering of the sectors and the octilinear directions relative to vertex u, e.g., $\mathrm{sec}_u(v) = 5$

Recall that $\mathrm{sec}_u(v)$ is a constant, thus $\mathrm{sec}_u^{\mathrm{pred}}$ etc. are also constants, and there is no need to perform modulo operations in the MIP.

We now restrict $\mathrm{dir}(u, v)$ (which is also used in other parts of the MIP formulation, e.g., in Sect. 4.4) to the set $\{\mathrm{sec}_u^{\mathrm{pred}}(v), \mathrm{sec}_u^{\mathrm{orig}}(v), \mathrm{sec}_u^{\mathrm{succ}}(v)\}$. For example, in the situation depicted in Fig. 9, we want that $\mathrm{dir}(u, v) \in \{4, 5, 6\}$. At the same time we must make sure that the values of $\mathrm{dir}(u, v)$ and $\mathrm{dir}(v, u)$ correspond to opposite directions. This is expressed by the disjunction

$$\begin{aligned}
&(\mathrm{dir}(u, v) = \mathrm{sec}_u^{\mathrm{pred}}(v) \wedge \mathrm{dir}(v, u) = \mathrm{sec}_v^{\mathrm{pred}}(u)) \vee \\
&(\mathrm{dir}(u, v) = \mathrm{sec}_u^{\mathrm{orig}}(v) \wedge \mathrm{dir}(v, u) = \mathrm{sec}_v^{\mathrm{orig}}(u)) \vee \\
&(\mathrm{dir}(u, v) = \mathrm{sec}_u^{\mathrm{succ}}(v) \wedge \mathrm{dir}(v, u) = \mathrm{sec}_v^{\mathrm{succ}}(u)) .
\end{aligned} \tag{9}$$

To model disjunction (9) we apply the first half of the standard trick that we detailed in Sect. 4.1: we introduce binary variables $\alpha_{\mathrm{pred}}(u, v)$, $\alpha_{\mathrm{orig}}(u, v)$, $\alpha_{\mathrm{succ}}(u, v)$, and the constraint

$$\alpha_{\mathrm{pred}}(u, v) + \alpha_{\mathrm{orig}}(u, v) + \alpha_{\mathrm{succ}}(u, v) = 1 \tag{10}$$

for each edge $uv \in E$. The (unique) variable that takes value 1 in (10) will determine the part of disjunction (9) that evaluates to true and thus the direction in which edge uv is drawn. Now the following constraint defines $\mathrm{dir}(u, v)$.

$$\mathrm{dir}(u, v) = \sum_{i \in \{\mathrm{pred}, \mathrm{orig}, \mathrm{succ}\}} \mathrm{sec}_u^i(v) \cdot \alpha_i(u, v) . \tag{11}$$

Here the unique variable $\alpha_i(u, v)$ that equals 1 selects the sector $\mathrm{sec}_u^i(v)$ that is assigned to $\mathrm{dir}(u, v)$. Note that the constraint is indeed linear since $\mathrm{sec}_u^i(v)$ is a constant. The constraint for $\mathrm{dir}(v, u)$ is analogous.

We use the variables of type $\alpha_i(u, v)$ not only to define $\mathrm{dir}(u, v)$ (which at the same time constrains the relative position of adjacent vertices u and v in the output drawing), but we also use the α_i's to enforce octilinearity. For example, let $\mathrm{sec}_u^{\mathrm{pred}}(v) = 4$ as in Fig. 9. Then we introduce the following constraints for edge uv and $i = \mathrm{pred}$:

$$\begin{aligned}
y(u) - y(v) &\le M(1 - \alpha_{\mathrm{pred}}(u, v)) \\
-y(u) + y(v) &\le M(1 - \alpha_{\mathrm{pred}}(u, v)) \\
x(u) - x(v) &\ge -M(1 - \alpha_{\mathrm{pred}}(u, v)) + \ell_{uv} ,
\end{aligned} \tag{12}$$

where $\ell_{uv} > 0$ is the minimum length of edge uv according to rule (R3). Here, we apply the second half of the standard trick from Sect. 4.1. In this case the value of the "large constant" M depends on the coordinate range. For example, we can set $M = n$ if at the same time we force the output drawing to lie in the square $[0, n] \times [0, n]$. Observe that constraints (12) are equivalent to $y(u) = y(v)$ and $x(u) \ge x(v) + \ell_{uv}$ if $\alpha_{\mathrm{pred}}(u, v) = 1$. This is exactly what is needed for an edge pointing horizontally to the left.

The constraints for other edge directions and for the cases $i = \mathrm{orig}$ and $i = \mathrm{succ}$ are constructed analogously.

4.4 Optimizing relative position

To preserve as much of the overall appearance of the subway network as possible we have already restricted the edge directions to the set of the three octilinear directions closest to the input direction. Ideally, one wants to draw an edge uv using its nearest octilinear approximation, i.e., the direction where $\mathrm{dir}(u, v) = \mathrm{sec}_u(v)$. Nöllenburg and Wolff model (R5) by introducing a cost of 1 in case the layout does not use that direction.

For each edge uv they define as its cost a binary variable $\rho(uv)$ which is 0 if and only if $\mathrm{dir}(u, v) = \mathrm{sec}_u(v)$. This is modeled by

$$-M\rho(uv) \leq \mathrm{dir}(u, v) - \mathrm{sec}_u(v) \leq M\rho(uv)\,. \tag{13}$$

Now the cost for deviating from the original relative positions can simply be expressed as

$$\mathrm{cost}_{(R5)} = \sum_{uv \in E} \rho(uv)\,. \tag{14}$$

4.5 Speed-up techniques

A common feature of subway maps is that they tend to have a large number of degree-2 vertices on line sections between two interchange stations. As we have seen in Sect. 3, both Hong et al. [15] and Stott and Rodgers [30] contract all degree-2 vertices, define appropriate edge weights, apply their layout algorithms on the contracted graph, and then reinsert the degree-2 vertices. Nöllenburg and Wolff [25] modify this data-reduction trick by keeping up to two dummy vertices on each chain of degree-2 vertices. The rationale behind drawing the connection between the corresponding interchange vertices as a polyline with at most three segments is that this distorts the map less than insisting on no bends. Again, the original vertices are reinserted uniformly on their corresponding polylines. Nöllenburg and Wolff report that their experiments showed that this is a good compromise between flexibility of the drawing and size of the MIP model. Recall that the target function penalizes bends along lines so that in many cases bends at these special degree-2 vertices are in fact avoided.

The only part of their MIP formulation that needs a quadratic number of constraints (and variables) is the one that ensures planarity, which is a direct consequence of rule (R3). This is why the most urgent need is to reduce the number of these constraints. An immediate reduction is as follows. For a planar drawing of an embedded graph it suffices to require that non-incident edges of the *same* face do not intersect. This already guarantees that no two edges intersect except at common endpoints. So instead of introducing planarity constraints for *all* pairs of non-incident edges, it is enough to include them only for pairs of non-incident edges of the same face.

However, the number of these planarity constraints is often still far too high, see Sect. 4.7. Nöllenburg and Wolff observed that, on the one hand, only a small fraction of all possible intersections was relevant for the layout. On the other hand, it is not clear how to determine relevant edge pairs in advance. A way out is the *callback* function of the MIP optimizer CPLEX. Nöllenburg and Wolff exploit this as follows. Their initial MIP formulation does not contain any planarity constraints at all. Then, during the optimization process, planarity constraints are added on demand as follows. Whenever the optimizer returns a new (and better) feasible solution, a callback routine is notified. This routine interrupts the optimizer and checks externally for edge crossings in the layout that corresponds to the current feasible solution. If the layout contains pairs of intersecting edges, Nöllenburg and Wolff add only the planarity constraints corresponding to those pairs and continue the optimization. If the current layout is plane, it is stored. The user can decide whether or not to continue the search for even better solutions. Section 4.7 shows the advantages of this approach in terms of number of constraints and running time.

4.6 Label placement

Subway maps in practice are of little interest to a passenger unless all stations are labeled with their names. Labels may not intersect each other or overlap vertices and edges of the graph. In a sense, they compete with the network for space and for an aesthetic placement. Therefore the significant amount of space required by labels ought to be taken into account during the layout process. Both Stott and Rodgers [31] and Nöllenburg and Wolff [25] take this so-called graph-labeling approach [17].

Still, Stott and Rodgers decouple layout and label process to a certain extent in the following sense. During their incremental layout process they repeatedly pick a vertex and move it to a better neighboring grid position. Labels are not taken into account when deciding which vertex to pick. Instead they locally find a best labeling *after* moving a vertex with similar metrics and a similar choice as in the case of vertices. They report that the tightly-coupled approach of considering label positions when evaluating vertex positions "proved to be excessively slow" [31].

The MIP formulation of Nöllenburg and Wolff must by definition use the tightly-coupled approach, and faces the same problem with running time. To counteract this, Nöllenburg and Wolff model all labels for collapsed degree-2 or degree-1 vertices along an edge together. The individual vertex labels are then placed inside a parallelogram-shaped region that is attached to the corresponding edge. If the

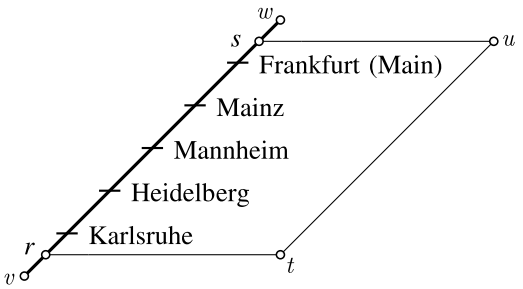


Fig. 10 Modeling vertex labels with a parallelogram-shaped region attached to edge vw

connection between two interchanges is modeled as a three-link path, the middle segment receives the edge label. The side length of the parallelogram matches the length of the longest vertex label. This enforces that all labels of stations along one edge are consistently placed on the same side of that edge, which is visually often more pleasing than an arbitrary mix of labels on both sides of the edge; see the soft part of rule (R8). Labels are restricted to be placed horizontally or, if the corresponding edge itself is horizontal, diagonally in z_1-direction. This keeps both the number of reading directions small and avoids unnecessary complexity in the model.

Nöllenburg and Wolff modify the given subway graph by adding new vertices and edges such that each parallelogram forms a new special face, see Fig. 10. Because the MIP creates a planar drawing of this extended subway graph, all labels can safely be placed inside the parallelograms. The new parts of the parallelograms can be seen as additional subway lines. They differ from the other subway lines only in that they can be embedded in two ways (instead of only one) and in that their shape is fixed. A label at an interchange station is modeled individually as a special edge of length equal to the label length, see e.g., station Karlsplatz in 11e. Binucci et al. [6] use a similar idea to label edges with rectangles in *orthogonal* graph drawings. In contrast to Nöllenburg and Wolff, Binucci et al. consider each edge and its label individually.

4.7 Experiments

In this section we report on additional experimental results with the mixed-integer programming approach of Nöllenburg and Wolff. Apart from the Sydney example that has been treated in Sect. 3, we picked two more examples (see [24] for more), namely Vienna and London. In both cases the input embedding was obtained by assuming straight-line edges between the stations. The MIPs were solved on the same machine as the Sydney example in Sect. 3.3. The size of the corresponding graphs is given in Table 2 (for the Sydney data, see Table 1) and the sizes

Table 2 Numbers n, m, m', and f of vertices, edges, multi-edges, and faces of the sample graphs, respectively

	Vienna (5 lines)				**London** (11 lines)			
G	n	m	m'	f	n	m	m'	f
original	90	96	96	8	308	361	441	55
contracted	44	50	50	8	186	239	307	55
labeled	98	117	60	21	453	550	396	99

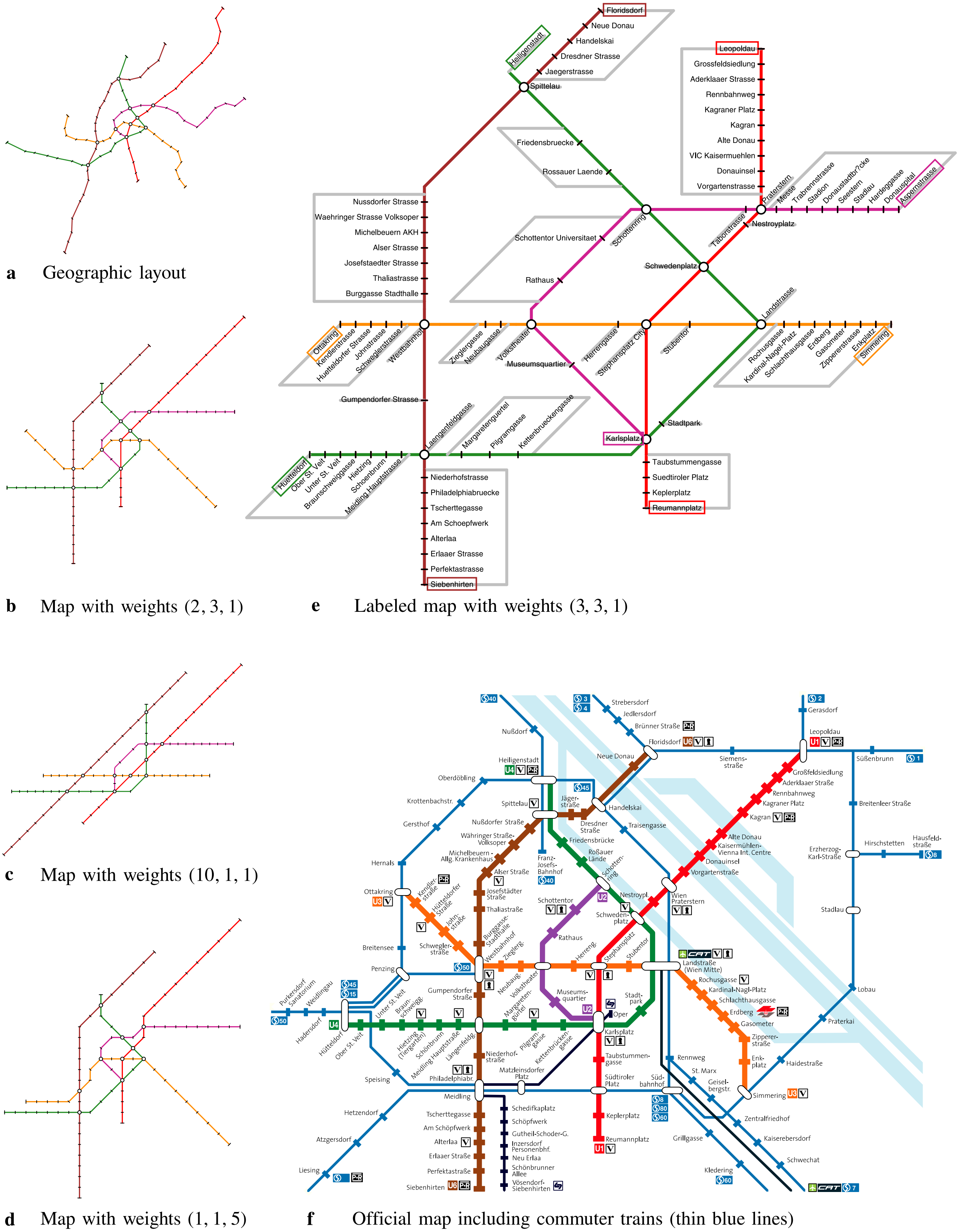
Table 3 Numbers of variables, constraints, and enforced non-intersecting edge pairs for the sample graphs

	unlabeled			**labeled**		
Vienna	all pairs	faces	none	all pairs	faces	none
variables	9,960	6,048	872	53,538	12,834	1,050
constraints	39,363	23,226	1,875	219,064	51,160	2,551
edge pairs	1,136	647	0	6,561	1,473	0
Sydney	all pairs	faces	none	all pairs	faces	none
variables	23,299	13,347	1,419	160,039	36,639	2,039
constraints	93,496	52,444	3,241	656,840	147,815	5,090
edge pairs	2,735	1,491	0	19,750	4,325	0
London	all pairs	faces	none	all pairs	faces	none
variables	227,535	53,487	4,063	1,204,343	118,879	5,655
constraints	930,863	212,925	9,041	4,958,294	480,755	13,706
edge pairs	27,934	6,178	0	149,863	14,153	0

of the MIPs are given in Table 3. There, the number of variables, constraints, and enforced non-intersecting edge pairs according to the planarity constraints is given, see Sect. 4.5. The columns *all pairs*, *faces*, and *none* contain the corresponding numbers for MIP formulations with planarity constraints for each pair of edges, for each pair of edges that lie on the same face, and without planarity constraints, respectively. The numbers of constraints and variables that were needed by the callback mechanism are bounded from below by column *none* and from above by column *faces*. These two columns also show that planarity is in fact responsible for 90%–95% of the constraints and variables.

The geographic layout of the subway system of Vienna is depicted in Fig. 11a. For the unlabeled drawing in 11b weights (2, 3, 1) were used for the soft constraints ((R4),(R5),(R6)) in the objective function (7). The drawing was obtained in 21 s as an intermediate feasible solution. The optimality gap for this solution was 19.7%. No additional planarity constraints were added by the callback function. Observe the influence of the soft constraints on the layout: there are no unnecessarily long edges (R6); the five subway lines only bend where geographically required and pass through interchange stations as straight as possible (R4); and, finally, the simplified edges tend to follow the original directions (R5).

Figures 11c and 11d show the influence of the soft constraints in an exaggerated fashion: increasing the weight for bends (R4) yields a drawing with as few bends as possible (see Fig. 11c), while increasing the weight for the network



a Geographic layout

b Map with weights (2, 3, 1)

c Map with weights (10, 1, 1)

d Map with weights (1, 1, 5)

e Labeled map with weights (3, 3, 1)

f Official map including commuter trains (thin blue lines)

Fig. 11 The Vienna subway network. Drawings **b**–**e** were produced with the MIP of Nöllenburg and Wolff using different weights in the objective function. The weights favor few bends (R4), good relative position (R5), and small network length (R6) in this order

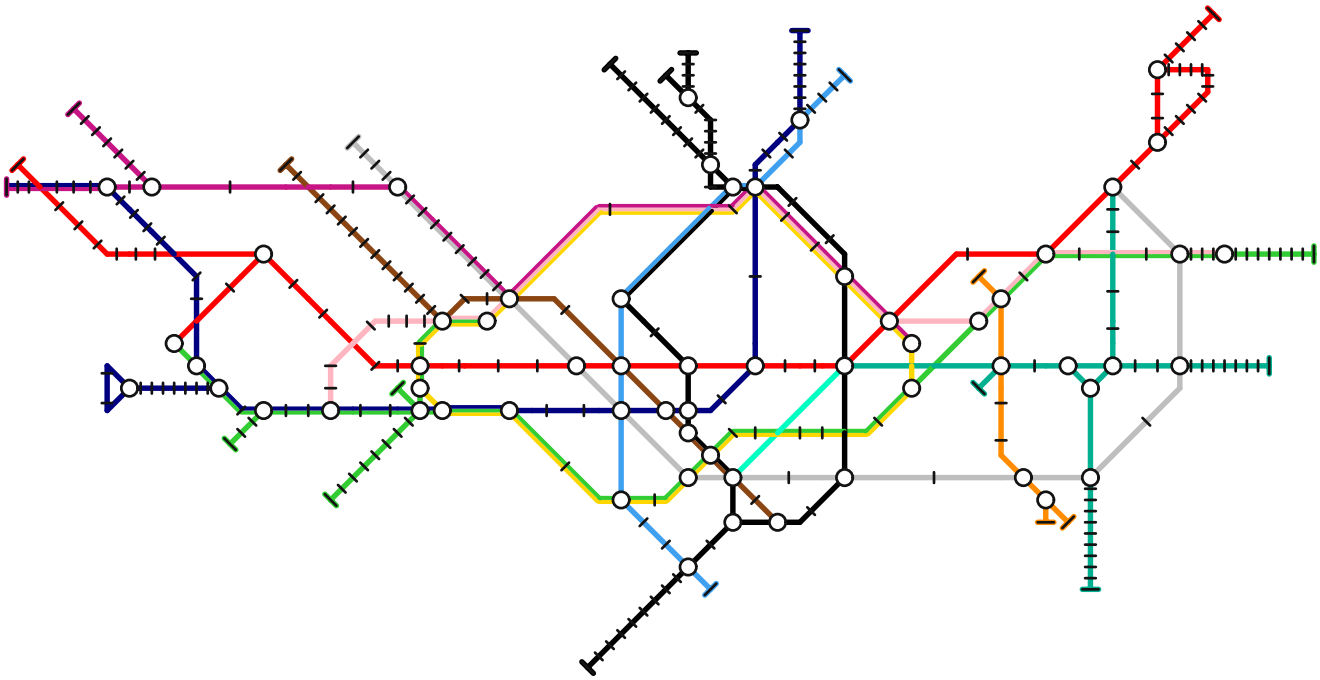


Fig. 12 Unlabeled London subway map produced by the MIP of Nöllenburg and Wolff

length (R6) yields a drawing where all edges but one span exactly one unit square (see Fig. 11d).

Figure 11e shows a labeled layout of the Vienna network with labels modeled as parallelogram-shaped faces. Here, the weights (3, 3, 1) were used for the objective function. To ensure planarity of the extended label graph, 183 edge pairs were forced to be non-intersecting, which added 5,856 constraints to the MIP. With 4 h and 7 min the computation time was much higher than for the unlabeled drawing. The optimality gap was 41.4%. The result in Fig. 11e shows that the MIP method is indeed capable of drawing labeled subway maps that are comparable to current hand-drawn maps. Some minor changes by a graphic designer would suffice to use such a map in practice. Figure 11f shows the official Vienna subway map. Note that the extension of the (violet) line U2 is still missing in the official map.

Finally, Fig. 12 shows an unlabeled layout of the London Underground network, one of the oldest and largest subway systems in the world. The weights for the objective function were (4, 1, 4) in this case and it took about 20 min to compute the layout with an optimality gap of 38.8%. The callback mechanism added 352 planarity constraints for 11 edge pairs to the MIP. The result is certainly not as sophisticated as the original London Tube Map[2], which has become a mental map of the city [26]. However, Fig. 12 does show that the MIP method has the potential to produce high-quality drawings even of large real-world subway networks. Unfortunately, the method did not succeed in finding a labeled layout for London due to the size of the corresponding MIP, see Table 3.

[2] See www.tfl.gov.uk/tube/maps/.

5 Complexity

In this section we discuss the computational complexity of drawing graphs with a given embedding. Before we turn to octilinear drawings, let us have a quick glance at orthogonal drawings, where all edges are drawn as rectilinear paths. Such drawings are very common for schematic diagrams in various applications such as flow charts or organigrams. The area of orthogonal graph drawing has been studied extensively; for an overview see the book of Di Battista et al. [10] or the survey of Eiglsperger et al. [11]. Tamassia's seminal work on orthogonal graph drawing has the following immediate consequence.

Theorem 1 (Tamassia [34]**).**
Let $G = (V, E)$ be a plane graph with maximum degree 4. Then there is an efficient algorithm that decides whether G can be drawn such that

1. *all edges are drawn as axis-parallel line segments,*
2. *the embedding of G is preserved, and*
3. *the drawing is plane.*

Tamassia's algorithm is more general in that it can layout any plane graph with maximum degree 4 such that the

edges are drawn as rectilinear paths, i.e., bends are allowed. Among all such layouts, the algorithm computes one with the minimum total number of bends. Note that the problem of finding a minimum-bend drawing immediately gets NP-hard [13] if G is not plane but planar, i.e., if the embedding of G is not fixed.

It is astonishing that adding two diagonal orientations increases the complexity of the subway-map layout problem drastically. Nöllenburg showed that one cannot expect to find an efficient algorithms that draws a given planar graph in a subway-map-like style.

Theorem 2 (Nöllenburg [24]**).**
OctilinearGraphDrawing *is NP-hard. In other words, given a plane graph $G = (V, E)$ with maximum degree 8, it is NP-hard to decide whether G can be drawn such that*

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

Since the proof is quite beautiful, we now give the details. The gadgets in the proof remind of the mechanical constructions that one can build with a Meccano or Märklin model construction kit. We hope that the figures help to seduce the reader to follow us through this proof since it exemplifies the idea behind so-called *reductions*. Reductions are a fundamental concept in theoretical computer science. In the following paragraph we explain how they are used to prove NP-hardness.

According to the definition of NP-hardness we have to show that every problem in the class $\mathcal{NP}$ (such as Satisfiability or GraphIsomorphism) can be *reduced* to our problem in polynomial time. In other words, if there were a polynomial-time algorithm for our problem, then *all* problems in the class $\mathcal{NP}$ could be solved in polynomial time. However, since other problems (such as Satisfiability or HamiltonianCircuit) are known to be NP-hard, it is enough to reduce *one* of those to our problem to show its hardness. This is what the following proof does.

Proof. (Nöllenburg [24]) The proof is by reduction from Planar 3-Sat, which is known to be NP-hard [19]. In an instance of Planar 3-Sat we are given a Boolean formula of a special type, and the task it to find an assignment of truth values to the variables in this formula such that the whole formula evaluates to *true*. Planar 3-Sat restricts the given Boolean formula φ in that

(a) φ must be in conjunctive normal form (CNF), i.e., a conjunction of disjunctions, e.g., $x_1 \wedge (\overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_3} \vee x_4)$,
(b) each disjunction (or *clause*) consists of exactly three literals, i.e., possibly negated variables, e.g., $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_3} \vee x_4)$, and
(c) the *variable-clause graph* H_φ is planar. The graph H_φ is bipartite; the vertices in one part of the bipartition represent the variables of φ, and the vertices in the other part represent the clauses of φ. Each clause is connected to the three variables that it contains. It is known [18] that the graph H_φ can be drawn as depicted in Fig. 13, i.e., all variables are placed on a horizontal line and the clauses are drawn as three-legged combs connecting from either above or below the variables.

Reducing OctilinearGraphDrawing to Planar 3-Sat means that we have to describe a polynomial-time transformation that maps a planar 3-CNF formula φ to a plane graph $G(\varphi)$ such that φ is satisfiable if and only if $G(\varphi)$ can be drawn octilinearly. Instead of considering φ itself, we take the planar embedded variable-clause graph H_φ as the object that will be transformed. Indeed, we construct the graph $G(\varphi)$ from two types of substructures in a way such that its overall structure resembles H_φ. We need one gadget to model the variables, i.e., a gadget that can be drawn in exactly two conformations representing the truth assignments of the respective variable. The second gadget will represent a clause of φ, so it has the shape of the combs in H_φ (recall Fig. 13) and is able to transmit the truth values of the three 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 evaluates to *true*. Thus, we can draw $G(\varphi)$ octilinearly if and only if φ is satisfiable.

The construction of the gadgets uses three basic building blocks that are depicted in Figs. 14 and 15. The main observation is that in the octilinear setting all non-degenerate triangles are isosceles right triangles since all angles must be multiples of 45°. So the degree of freedom when drawing an octilinear triangle with one side fixed consists in the choice of the vertex adjacent to the right angle, see Fig. 14a.

Now, we can combine several triangles to form compound structures such as the square block in Fig. 14b. The underlying plane graph with five vertices and four triangular faces has the property that Fig. 14b is the only way of drawing it octilinearly. The reason is that vertex E is incident to the four triangular faces and thus the four angles adjacent to E must sum to 360°. But this is the case only if the

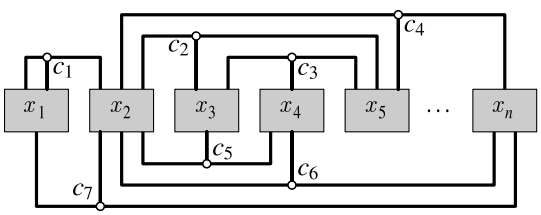
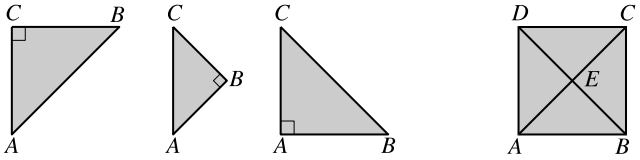


Fig. 13 Example of a planar variable-clause graph



a **b**

Fig. 14 Basic building blocks. A triangle with fixed side AC has exactly three different octilinear realizations (**a**). The square block in **b** has a fixed octilinear shape

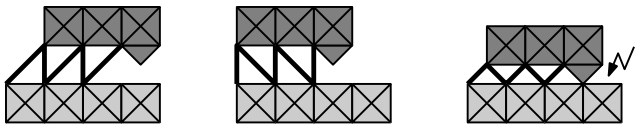


Fig. 15 The translational joint

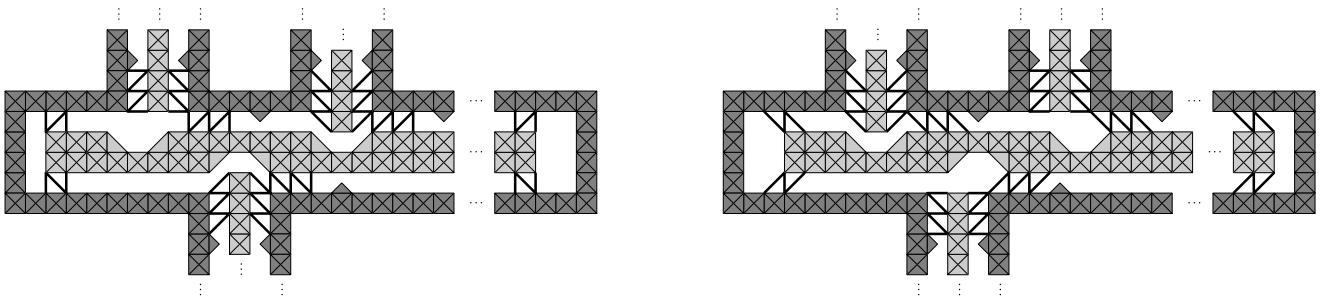
four right angles of the triangles are adjacent to E, which defines the shape of the graph. Obviously, larger rigid structures can be constructed by attaching these square blocks side-by-side.

The third building block models a translational joint between two rigid components. To this end we connect two bars made out of square blocks by four adjacent triangles as depicted in Fig. 15. These triangles admit exactly the three octilinear realizations of the combined structure that are shown in Fig. 15. To rule out the rightmost realization we add a *spacer* triangle to the upper bar. Now, the layout on the right-hand side violates planarity as the spacer touches the other bar, while the other two drawings remain valid. Note that in this structure all square blocks necessarily have the same unit size and that the distance between the two bars also equals one unit.

Now we can construct more complex structures that serve as gadgets for variables and clauses of φ. It is important that all parts of these gadgets are connected in such a way that the side lengths of the square blocks involved are equal. This is ensured by connecting square blocks side-by-side or by using the translational joint to connect square blocks. 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, the gadget must be able to transmit its truth value to the clauses containing this variable, depending on whether the variable is negated in the respective clause or not. A sample variable gadget is shown in Fig. 16. The main part of this gadget is a large horizontal *variable bar* in the middle of the construction made up of square blocks and containing some dents. It is framed by a box of square blocks with upward and downward ports that will be connected to the clause gadgets. The variable bar is always in one of two positions that are horizontally one unit apart, see Fig. 16. Let the left position represent the 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 their tunnels by translational joints and hence can move upwards and downwards. The literal bars can only be moved towards the variable bar if there is a dent aligned with the tunnel opening. If there is no dent aligned 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 is in the position of Fig. 16a. 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 as in Fig. 16b. Consequently, only those literal bars corresponding to literals that evaluate to *true* can be fitted into their dents. This fact is required for constructing the clause gadget. Note that all parts of the structure are connected such that they use the same square block size and thus all vertices are placed on a grid.

Second, we describe the clause gadget. It must be constructed such that it is planar if and only if at least one literal of that clause is *true*, i.e., one of the literal bars in the tun-



a Configuration corresponding to true **b** Configuration corresponding to false

Fig. 16 The variable gadget

nels connected to the variables fits into its dent. In other words, the gadget cannot be drawn correctly if there is pressure from all three literals bars. Now we describe the clause gadget, as shown in Fig. 17, in detail.

The clause gadget has the shape of a three-legged comb just as in the variable-clause graph H_φ, recall Fig. 13. 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 of the gadget. At this turn the vertical pressure from the variable is transformed into horizontal pressure. This can be seen in Fig. 17a. The left literal bar is in its 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 its 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 this gadget is a flexible *switch* in the center of the gadget that is able to select a satisfied literal if there is one. As a whole, this switch can shift vertically by one unit due to four joints fixing it to the walls of the gadget. In Fig. 17a it is in its lower position and in Figs. 17b and 17c 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 moved to the left and to the right independently using two standard translational joints. 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 can be seen in Fig. 17c, where the left literal bar and the switch bar touch each other and thus violate planarity. Hence, the whole gadget can be drawn octilinearly as long as at least one literal evaluates to *true* and the switch is shifted towards this very literal. However, if all literals are *false* and hence the respective bars are shifted into 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.

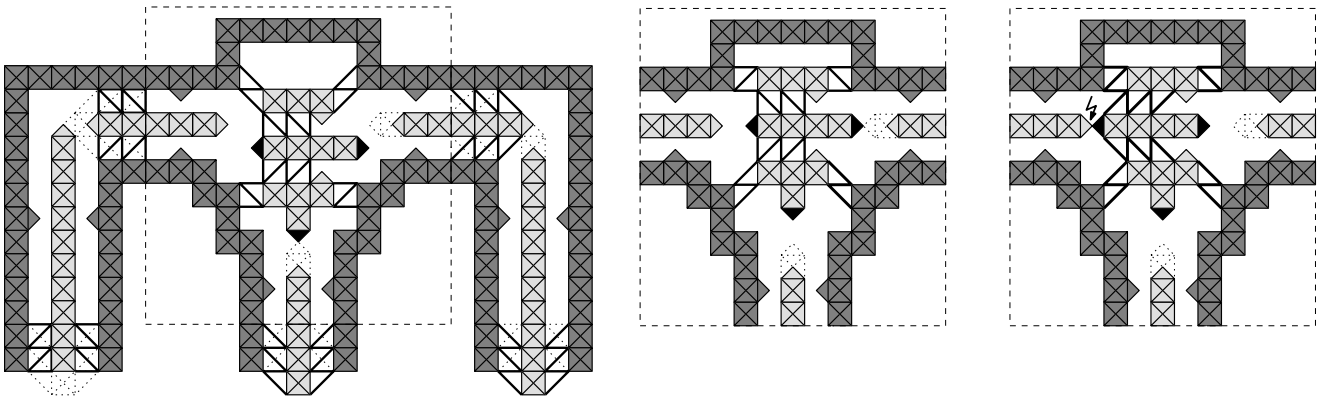
With these two gadgets for variables and clauses we can construct the whole graph $G(\varphi)$ by connecting the literal tunnels of the clause gadgets to their ports in the respective variable gadgets. The resulting graph is planar since the input variable-clause graph H_φ is planar. We choose the input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

To conclude the proof, 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 octilinearly such that 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. If, however, φ is not satisfiable then for each variable assignment there is at least one clause that evaluates to *false*. This means that $G(\varphi)$ cannot be drawn octilinearly because in the corresponding clause gadget all literal bars are pushed towards the center of this clause 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. □

We have just seen that OctilinearGraphDrawing is a hard problem. Now we set out to show that among the hard problems it is on the easy side. We prove that OctilinearGraphDrawing lies in $\mathcal{NP}$, i.e., there is a *nondeterministic* polynomial-time algorithm for it. This means that it is "easier" than, say, QuantifiedBooleanFormula, which is PSPACE-hard, and much easier than the halting problem, which is undecidable.

Corollary 1. OctilinearGraphDrawing *is NP-complete.*



a The second literal is selected by the switch **b** The third literal is selected **c** The first literal is selected

Fig. 17 The clause gadget. In each of the subfigures the literal bar in the left leg transmits *false* while the other two bars transmit *true*. Alternative literal bar positions are indicated with dotted lines. In **c** the switch selects the first literal. This violates planarity at the flash symbol

Proof. According to Theorem 2, the problem OctilinearGraphDrawing is NP-hard. To show the NP-completeness of OctilinearGraphDrawing it remains to prove that the problem lies in $\mathcal{NP}$. A common way to show that a problem Π lies in $\mathcal{NP}$ is to specify for each instance I of Π a finite set S_I such that if I is a "yes"-instance, S_I contains a witness W for this and W can be verified deterministically in polynomial time. Then a non-deterministic algorithm for Π consists of "guessing" an element of S_I and then verifying it. In our case we could let S_I be the set of all straight-line drawings of a given planar graph G. Then verification would be easy, since for such a drawing one can check efficiently whether all edges are octilinear, the embedding is preserved, and the drawing is planar. However, the cardinality of S_I would not necessarily be finite, even if scaling and rotating were factored out.

Instead, we utilize the MIP formulation of Sect. 4 restricted to the part that models the hard constraints (R1)–(R3). This MIP has a solution if and only if there is an embedding-preserving octilinear planar drawing of the input graph G, i.e., if and only if the conditions of Theorem 2 are met.

In this MIP all integer variables are bounded by small constants and the only non-integer variables are the vertex coordinates. Now we can guess an assignment of the integer variables, which basically corresponds to guessing edge directions. To decide whether such an assignment is feasible, we simply solve the linear program (LP) that corresponds to an instance of the MIP in which all integer variables have been set according the assignment that we guessed. The LP can be solved in $O(N^{3.5}L^2)$ time by Karmakar's interior-point method [16], where N is the number of constraints and L is the number of input bits. The LP consists of $N = O(n^2)$ constraints and variables. Its coefficients are small integers each of which requires only a constant number of bits. Hence, solving the LP is polynomial in terms of the size of the input graph, and OctilinearGraphDrawing is in $\mathcal{NP}$. □

6 Conclusion and future work

We have surveyed three methods for drawing subway networks; a spring embedder by Hong et al. [15], a hill climber by Stott and Rodgers [30], and a MIP-based method by Nöllenburg and Wolff [25]. The medium-size CityRail network of Sydney is the only network that was drawn by each of the methods. We have compared the drawings in the corresponding publications and have evaluated them using a list of rules. The method of Hong et al. was by far the fastest, but in terms of quality inferior to the other two methods. The method of Nöllenburg and Wolff was quite fast on this particular example, but since it relies on a MIP solver, its running time is difficult to predict. The strength of their method is the fact that the hard constraints – including octilinearity – are fulfilled by construction. The hill-climber based method of Stott and Rodgers managed to force all edges but a few to octilinear directions, but it introduced more bends than the MIP-based method.

The pros and cons of the two slower methods are the following. When using hill climbing, on the one hand the optimization may be stopped at any time and yields *some* result. On the other hand, the process usually gets stuck in local optima. Then it fails to improve even when given a lot of time. Another minus is that the multi-criteria optimization does not guarantee octilinearity.

In contrast, the MIP approach by construction will only find drawings that fulfill all hard constraints. For unlabeled drawings the MIP solver usually quickly generates good intermediate solutions, but proving their optimality can take a long time. However, in practice it is usually not worth waiting for the optimal solution (in terms of the objective function). Instead, any good feasible solution will do. Recall that the objective function is just an attempt to mathematically formulate the aesthetic quality of a drawing and hence the optimal solution is not necessarily more pleasing (for a human viewer) than some close-by solutions.

For labeled subway maps the MIPs become significantly larger in terms of numbers of variables and constraints, and hence their optimization takes much more time. The results for examples like Vienna and Sydney look promising, but the MIP of Nöllenburg and Wolff did not succeed to find a labeled drawing for the large-scale network of London. Thus further heuristic speed-ups for combining the drawing and labeling of subway maps are required. One could, for example, soften the requirement that labels may not overlap edges of the network. Instead one could punish such overlaps by the objective function and hope that most of them will disappear due to optimization. One could also precompute simple parts of the network including their labels, e.g., long suburban lines that lie completely in the outer face, assuming that this would speed up the optimization of the denser and more difficult downtown area [28]. Possibly the path-simplification algorithm of Merrick and Gudmundsson [20] (see Sect. 3.4) could be integrated into a global optimization procedure that forbids intersection of independent paths.

Which of the two slower methods – hill climbing versus MIP – yields nicer drawings may depend on personal preferences. Both succeed in drawing unlabeled networks in a quality comparable to maps drawn by graphic designers. This indicates that our list of rules indeed models similar layout criteria as those applied by professionals. A user study would help to clarify in how far these criteria do in fact support subway passengers in quickly making the right decisions.

One may ask why maps drawn by graphic designers appear still somewhat more pleasing and elegant than the best

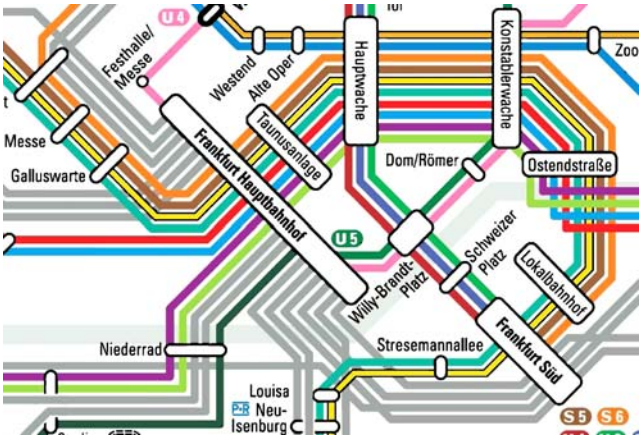


Fig. 18 Clipping of the official map of the Frankfurt public transportation network, which has many parallel lines and uses octilinearly oriented rectangles to mark stations

automatically generated maps. We think that there are two main reasons. First, the automatically generated drawings we see today are the result of academic feasibility studies, not the output of professional tools. Therefore, they lack the finishing touch that professional graphic designers apply to their drawings: nice fonts and colors, rounded bends, special symbols for interchange stations, line breaks in labels etc. Second, and more important, a professional graphic designer uses background knowledge that is not available to the current algorithms: (s)he sees symmetries or knows underlying structure (such as the circular lines in London or Moscow), and can stress or otherwise take advantage of these in the layout process. In a way, a good layout of a complicated subway system is (still!) a piece of art, and thus may be out of reach for complete automation. Maybe this is the beginning of a new field: computational aesthetics or computational design?

We close with a concrete problem that the current MIP formulation ignores, namely parallel subway lines. These appear quite often, especially in German subway networks (see Fig. 18), but also elsewhere (see Fig. 3b). With the increasing multiplicity of edges it becomes difficult to visually keep the different colors of the lines apart. This phenomenon can be counteracted by increasing line thickness and interline distances. This in turn would force incident stations to become larger. Ideally, the sizes of vertices and the thickness of edges would be integrated into the model. This would also remove the current restriction (of 8) on the maximum vertex degree of the underlying subway graph. Some of this work can already be seen in the MIP formulation of Binucci et al. [6], where vertices and labels are represented by axis-parallel rectangles – for drawing graphs orthogonally.

Acknowledgement I thank Seok-Hee Hong and Herman Haverkort for interesting discussions during their respective visits to Karlsruhe in 2004 and Damian Merrick for providing me with the drawings in Fig. 4. I am indebted to Martin Nöllenburg for the material in Sect. 5, for pointing me to Fig. 1, and for intensive discussions, especially about the remaining insufficiencies of automatically generated subway maps. Thanks to Dorothea Wagner for insisting that I should investigate this beautiful topic and for a hardcopy of Fig. 2. Thanks to the publisher O'Reilly for permission to use Fig. 2. Finally I thank the anonymous referees of this article for their detailed and very helpful comments.

References

1. Avelar S, Müller M (2000) Generating topologically correct schematic maps. In: Proc. 9th Internat. Sympos. Spatial Data Handling (SDH'00), pages 4a.28–4a.35
2. Barkowsky T, Latecki LJ, Richter KF (2000) Schematizing maps: Simplification of geographic shape by discrete curve evolution. In: Freksa C, Brauer W, Habel C, Wender KF (Eds) Proc. Spatial Cognition II – Integrating abstract theories, empirical studies, formal models, and practical applications. Lect Notes Artific Intell 1849:41–53
3. Bekos M, Kaufmann M, Potika K, Symvonis A (2008) Line crossing minimization on metro maps. In: Hong SH, Nishizeki T (Eds) Proc. 15th Internat. Sympos. Graph Drawing (GD'07), Lect Notes Comput Sci 4875:231–242. Springer-Verlag
4. Benkert M, Nöllenburg M, Uno T, Wolff A (2007) Minimizing intra-edge crossings in wiring diagrams and public transport maps. In: Kaufmann M, Wagner D (Eds) Proc. 14th Internat. Sympos Graph Drawing (GD'06), Lect Notes Comput Sci 4372:270–281
5. Bertault F (2000) A force-directed algorithm that preserves edge-crossing properties. Inf Proc Lett 74(1):7–13
6. Binucci C, Didimo W, Liotta G, Nonato M (2005) Orthogonal drawings of graphs with vertex and edge labels. Comput Geom Theory Appl 32(2):71–114
7. Brandes U, Eiglsperger M, Kaufmann M, Wagner D (2002) Sketch-driven orthogonal graph drawing. In: Kobourov SG, Goodrich MT (Eds) Proc. 10th Internat. Sympos. Graph Drawing (GD'02), Lect Notes Comput Sci 2528:1–11
8. Cabello S, de Berg M, van Kreveld M (2005) Schematization of networks. Comput Geom Theory Appl 30(3):223–238
9. Cabello S, van Kreveld M (2003) Approximation algorithms for aligning points. In: Proc. 19th Annu. ACM Sympos. Comput. Geom. (SoCG'03), pp 20–28
10. Di Battista G, Eades P, Tamassia R, Tollis IG (1999) Graph Drawing. Prentice Hall, Upper Saddle River, NJ
11. Eiglsperger M, Fekete SP, Klau GW (2001) Orthogonal graph drawing. In: Kaufmann M, Wagner D (Eds) Drawing Graphs: Methods and Models, Lect Notes Comput Sci 2025(6)121–171
12. Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129–1164
13. Garg A, Tamassia R (2001) On the computational complexity of upward and rectilinear planarity testing. SIAM J Comput 31(2):601–625
14. Garland K (1994) Mr Beck's Underground Map. Capital Transport Publishing, Harow Weald, Middlesex, England
15. Hong SH, Merrick D, do Nascimento HAD (2006) Automatic visualisation of metro maps. J Vis Lang Comput 17(3):203–224
16. Karmakar N (1984) A new polynomial-time algorithm for linear programming. Combinatorica 4(4):373–395

17. Klau GW, Mutzel P (1999) Combining graph labeling and compaction. In: Kratochvíl J (Ed) Proc. 7th Internat. Sympos. Graph Drawing (GD'99), Lect Notes Comput Sci 1731:27–37
18. Knuth DE, Raghunathan A (1992) The problem of compatible representatives. SIAM J Discr Math 5(3):422–427
19. Lichtenstein D (1982) Planar formulae and their uses. SIAM J Comput 11(2):329–343
20. Merrick D, Gudmundsson J (2006) Path simplification for metro map layout. In: Kaufmann M, Wagner D (Eds) Proc. 14th Internat. Sympos. Graph Drawing (GD'06), Lect Notes Comput Sci 4372:258–269
21. Morrison A (1996) Public transport maps in western european cities. Cartogr J 33(2):93–110
22. Nesbitt KV (2004) Getting to more abstract places using the metro map metaphor. In: Proc. 8th Internat. Conf. Inform. Visualisation (IV'04), IEEE Computer Society pp. 488–493
23. Neyer G (1999) Line simplification with restricted orientations. In: Dehne FK, Gupta A, Sack JR, Tamassia R (Eds) Proc. 6th Internat. Workshop Algorithms Data Struct. (WADS'99), Lect Notes Comput Sci 1663:13–24
24. Nöllenburg M (2005) Automated drawings of metro maps. Technical Report 2005-25, Fakultät für Informatik, Universität Karlsruhe. Available at http://www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2005/25
25. Nöllenburg M, Wolff A (2006) A mixed-integer program for drawing high-quality metro maps. In: Healy P, Nikolov NS (Eds) Proc. 13th Internat. Sympos. Graph Drawing (GD'05), Lect Notes Comput Sci 3843:321–333
26. Ovenden M (2003) Metro Maps of the World. Capital Transport Publishing, Harrow Weald, Middlesex, England
27. Polatschek K (2006) Die Schönheit des Untergrundes. Frankfurter Allgemeine Sonntagszeitung 28, 16 July. Available via http://fazarchiv.faz.net
28. Roberts M (2007) Personal email communication
29. Sandvad ES, Grønbæk K, Sloth L, Knudsen JL (2001) A metro map metaphor for guided tours on the Web: the Webvise Guided Tour System. In: Proc. 10th Internat. World Wide Web Conf. (WWW'01), pp. 326–333. ACM Press
30. Stott J, Rodgers P (2004) Metro map layout using multicriteria optimization. In: Proc. 8th Internat. Conf. Inform. Visualisation (IV'04), pp. 355–362. IEEE Comput Soc
31. Stott JM, Rodgers P (2005) Automatic metro map design techniques. In: Proc. 22nd Internat. Cartographic Conf. (ICC'05), La Coruña, Spain
32. Stott JM, Rodgers P, Burkhard RA, Meier M, Smis MTJ (2005) Automatic layout of project plans using a metro map metaphor. In: Proc. 9th Internat. Conf. Inform. Visualisation (IV'05), pp. 203–206
33. Sydney CityRail
34. Tamassia R (1987) On embedding a graph in the grid with the minimum number of bends. SIAM J Comput 16(3): 421–444
35. Ware JM, Anand S, Taylor GE, Thomas N (2006) Automated production of schematic maps for mobile applications. Trans in GIS 10(1):25–42
36. X Initiative. http://www.xinitiative.org