

On Metro-Line Crossing Minimization¹

*Evmorfia Argyriou*¹ *Michael A. Bekos*¹ *Michael Kaufmann*²
*Antonios Symvonis*¹

¹School of Applied Mathematics & Physical Sciences,
National Technical University of Athens, Greece

²Institute for Informatic,
University of Tübingen, Germany

Abstract

We consider the problem of drawing a set of simple paths along the edges of an embedded underlying graph $G = (V, E)$ so that the total number of crossings among pairs of paths is minimized. This problem arises when drawing metro maps, where the embedding of G depicts the structure of the underlying network, the nodes of G correspond to train stations, an edge connecting two nodes implies that there exists a railway track connecting them, whereas the paths illustrate the metro lines connecting terminal stations. We call this the *metro-line crossing minimization problem (MLCM)*. We examine several variations of the problem for which we develop algorithms that yield optimal solutions.

Submitted: November 2008	Reviewed: April 2009	Revised: April 2009	Accepted: November 2009
	Final: December 2009	Published: January 2010	
Article type: Regular paper		Communicated by: I. G. Tollis and M. Patrignani	

E-mail addresses: fargyriou@math.ntua.gr (Evmorfia Argyriou) mikebekos@math.ntua.gr (Michael A. Bekos) mk@informatik.uni-tuebingen.de (Michael Kaufmann) symvonis@math.ntua.gr (Antonios Symvonis)

¹Preliminary versions of this work have appeared in [5] and [3].

1 Introduction

Metro maps or public transportation networks are quite common in our daily life and familiar to most people. The visualization of such maps takes inspiration from the fact that the passengers riding the trains are not too concerned about the geographical accuracy of the train stations, but they are more interested in how to get from one station to another and where to change trains. Therefore, almost all metro maps look like electrical schematics (i.e., circuit boards, usually orthogonal; see Figure 1), on which all stations are almost equally spaced, rather than geographic maps (see Figure 2).



Figure 1: An illustration of the metro map of Washington DC. Taken from <http://www.airwise.com/airports/us/IAD/images/metromap.gif>

In general, a metro map can be modeled as a tuple (G, L) , which consists of a connected graph $G = (V, E)$, referred to as the *underlying network*, and a set L of simple paths on G . The nodes of G correspond to train stations, an edge connecting two nodes implies that there exists a railway track connecting them, whereas the paths illustrate the metro lines connecting terminal stations. Then, the process of constructing a metro map consists of a sequence of steps. Initially, one has to draw the underlying network nicely. Then, the lines have to be properly added into the visualization and, finally, a labeling of the map

1.1 Problem Definition

The input of the metro-line crossing minimization problem consists of a connected, embedded, planar graph $G = (V, E)$ and a set $L = \{l_1, l_2 \dots l_{|L|}\}$ of simple paths on G , called *lines*. We refer to G as the underlying network and to the nodes of G as *stations*. We also refer to the endpoints of each line as its *terminals*. The stations are represented as particular shapes (usually as rectangles but in general as polygons). The sides of each station that each line may use to either “enter” or “leave” the station are also specified as part of the input. Motivated by the fact that a line cannot make a 180° turn within a station, we do not permit a line to use the same side of a station to both “enter” and “leave”.

The output of the MLCM problem specifies an ordering of the lines at each side of each station so that the number of crossings among pairs of lines is minimized.

Formally, each line $l \in L$ consists of a sequence of edges $e_1 = (v_0, v_1), \dots, e_d = (v_{d-1}, v_d)$. Stations v_0 and v_d are the terminals of line l . Equivalently, we say that l *terminates* or *has terminals at* v_0 and v_d . By $|l| = d$ we denote the length of line l .

One can define several variations of the MLCM problem based on the type of the underlying network, the location of the crossings and/or the location of the terminals (refer to Figure 1). As already stated, each line that traverses a station u has to touch two of the sides of u at some points (one when it “enters” u and one when it “leaves” u). These points are referred to as *tracks* (see the white colored bullets on the boundary of each station in Figure 3b). In general, we may permit tracks on all sides of each station, i.e., a line that traverses a station may use any side of it to either “enter” or “leave” (see Figure 3a). In the case where the stations are represented as rectangles, this model is referred to as the *4-side* model. In the general case where the stations are represented as polygons of at most k sides, this model is referred to as the *k-side* model. A more restricted model, referred to as the *2-side* model, is the one where i) the stations are represented as rectangles and ii) all lines that traverse a station may use only its left and right side (see Figure 3b).

A particularly interesting case that arises under the 2-side model and concerns the location of the line terminals at the nodes is the one where the lines that terminate at a station occupy its topmost and bottommost tracks, in the following referred to as *top* and *bottom station ends*, respectively. The remaining tracks on the left and right side of the station are referred to as *middle tracks* and are occupied by the lines that pass through the station. Figure 4 illustrates the notions of top and bottom station ends and middle tracks on the left and right side of a station (solid lines correspond to lines that terminate, whereas the dashed lines correspond to lines that go through the station). Based on the above, we define the following two variants of the MLCM problem:

- (a) *The MLCM problem with terminals at station ends* (MLCM-SE), where we ask for a drawing of the lines along the edges of G so that (i) all lines

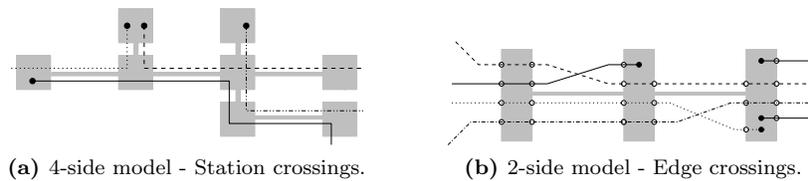


Figure 3: The underlying network is the gray colored graph. We have used different types of lines to denote different lines.

terminate at station ends and (ii) the number of crossings among pairs of lines is minimized.

- (b) *The MLCM problem with terminals at fixed station ends (MLCM-FixedSE)*, where all lines terminate at station ends and the information whether a line terminates at a top or at a bottom station end in its terminal stations is specified as part of the input. We ask for a drawing of the lines along the edges of G so that the number of crossings among pairs of lines is minimized.

A further refinement of the MLCM problem concerns the location of the crossings among pairs of lines. If the relative order of two lines changes between two consecutive stations, then the two lines must intersect between these stations (see Figure 3b). We call this an *edge crossing*. Opposed to an edge crossing, a *station crossing* occurs inside a station. In order to avoid the case where a great number of crossings takes place in the interior of a station, which unavoidably leads to cluttered drawings, we seek to avoid station crossings whenever this is possible. This seems quite important especially in the case where the size of the stations is not quite large. Additionally, station crossings are further obscured by the presence of the stations themselves, while the line crossings that occur along the edges of the underlying network are not negatively influenced by the edges, since the edges are rarely drawn on the map. However, it is not always feasible to avoid station crossings. For instance, if two lines share a common path of the underlying network (see the dotted and dashed-dotted lines in Figure 3b), then we could either place their crossing along an edge or in the interior of a station of their common path. In this case we would prefer the edge crossing. However, it is not always feasible to avoid station crossings. To realize that, refer to Figure 3a, where the two crossing lines share only a common station and they have to cross (the dashed-dotted line traverses the station from its top side to its bottom side, where the dashed one from left to right). In this case, their crossing will inevitably occur in the interior of their common station. Such crossings (i.e., that cannot be avoided) are referred to as *unavoidable* crossings.

1.2 Previous Work and Our Results

The problem of drawing a graph with a minimum number of crossings has been extensively studied in the graph drawing literature. We refer to the monographs of Di Battista et al. [8] and Kaufmann and Wagner [13]. In the problems we

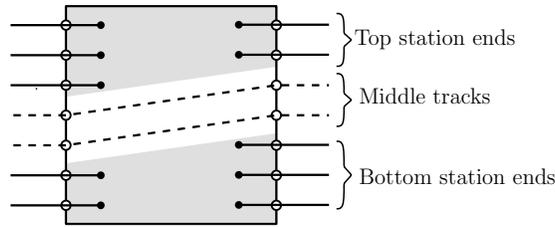


Figure 4: Station ends and middle tracks.

study, however, we assume that the underlying graph has already received an embedding and we seek to draw the lines along the graph’s edges so that the number of crossings among pairs of lines is minimized.

The MLCM problem was recently introduced by Benkert et al. [6]. In their work, they proposed a dynamic-programming based algorithm that runs in $O(n^2)$ time for the so-called *one-edge layout problem*, which is defined as follows: Given a graph $G = (V, E)$ and an edge $e = (u, v) \in E$, let L_e be the set of lines that traverse e . The set L_e is divided into three subsets L_u , L_v , and L_{uv} . Set L_u (L_v) consists of the lines that traverse u (v) and terminate at v (u). Set L_{uv} consists of the lines that traverse both u and v and neither terminate at u nor at v . The lines for which u is an intermediate station, i.e., $L_{uv} \cup L_u$, enter u in a predefined order S_u . Analogously, the lines for which v is an intermediate station, i.e., $L_{uv} \cup L_v$, enter v in a predefined order S_v . The number of pairs of crossing lines is then determined by inserting the lines of L_u into the order S_v and by inserting the lines of L_v into the order S_u . The task is to determine an insertion order so that the number of pairs of crossing lines is minimized. Benkert et al. [6] do not address the case of larger graphs; they leave as an open problem the case where the lines that terminate at a station occupy its station ends.

Bekos et al. [5] proved that MLCM-FixedSE problem can be solved in $O(|V| + \log \Delta \sum_{l \in L} |l|)$, in the case where the underlying network was a tree of degree Δ . Extending the work of Bekos et al., Asquith et al. [4] proved that the MLCM-FixedSE problem was also solvable in polynomial time in the case where the underlying network was an arbitrary planar graph. The time complexity of their algorithm was $O(|E|^{5/2}|L|^3)$. They also proposed an integer linear program which solves the MLCM-SE problem.

A closely related problem to the one we consider is the problem of drawing a metro map nicely, known as the *metro map layout problem*. Hong et al. [11] implemented five methods for drawing metro maps using modifications of spring-based graph drawing algorithms. Stott and Rodgers [18] approached the problem by using a multi-criteria optimization based on hill climbing. The quality of a layout was a weighted sum over five metrics that were defined for evaluating the niceness of the resulting drawing. Nöllenburg and Wolff [16] specified the niceness of a metro map by listing a number of hard and soft constraints and proposed a mixed-integer program which always determines a

drawing that fulfills all hard constraints (if such a drawing exists) and optimizes a weighted sum of costs corresponding to the soft constraints.

This paper is structured as follows: In Section 2, we show that the MLCM-SE problem is *NP*-complete, even in the case where the underlying network is a path. In Section 3, we present a polynomial time algorithm that runs in $O((|E| + |L|^2)|E|)$ time for the MLCM problem under the k -side model, assuming that the line terminals are located at stations of degree one. To the best of our knowledge no results are currently known regarding this model. In Section 4, we present a faster algorithm for the special case of 2-side restriction. The time complexity of the proposed algorithm is $O(|V||E| + \sum_{l \in L} |l|)$. We further prove that the MLCM-FixedSE problem can be reduced to this restricted model resulting in an algorithm that drastically improves the running time of the algorithm of Asquith et al. [4] from $O(|E|^{5/2}|L|^3)$ to $O(|V||E| + |V||L|)$. We conclude in Section 5 with open problems and future work.

2 The MLCM-SE Problem

In this section, we study the metro-line crossing minimization problem assuming that the underlying network G is a path and its nodes are restricted to lie on a horizontal line. We consider the 2-side model where each line enters (exists) an internal station on its left (right) side. Then, assuming that there exist no restrictions on the location of the line terminals at the nodes, it is easy to see that there exist solutions without any crossing among lines. In fact, using a simple reduction from the interval graph coloring problem [10], we can determine a solution, which also minimizes the number of tracks used at each individual node. So, in the rest of this section, we further assume that the lines that terminate at a station occupy its top and bottom station ends. In particular, we consider the MLCM-SE problem on a path. Since the order of the stations is fixed as part of the input of the problem, the only remaining choice is whether each line terminates at the top or at the bottom station end in its terminal stations. In the following we show that under this assumption, the problem of determining a solution so that the total number of crossings among pairs of lines is minimized is *NP*-complete. Our proof is based on a reduction from the *fixed linear crossing number problem* [14].

Definition 1 *Given a simple graph $G = (V, E)$, a linear embedding of G is an embedding of G in which the nodes of V are placed on the x -axis and the edges are drawn as semicircles either above or below the x -axis.*

Definition 2 *A node ordering (or a node permutation) of a graph G is a bijection $\delta : V \rightarrow \{1, 2, \dots, n\}$, where $n = |V|$. For each pair of nodes u and v , with $\delta(u) < \delta(v)$ we shortly write $u < v$.*

Note that the crossing number of a linear embedding is determined by the node ordering. However, Masuda et al. [14] proved that, even if the node ordering is fixed, it is *NP*-hard to determine a linear embedding of a given graph

with the minimum number of crossings. The latter problem is referred to as *fixed linear crossing number problem*. Observe that the computational difficulty of this problem merely lies in determining, for each edge, whether to place it above or below the x -axis.

Theorem 1 *The MLCM-SE problem on a path is NP-complete.*

Proof: We will prove that, given a positive integer $c \in \mathbb{Z}^+$, the problem of finding a solution of the MLCM-SE problem on a path with total number of crossings no more than c , is NP-complete. Membership in NP follows from the fact that a nondeterministic algorithm needs only to guess an ordering of the lines at the left and the right side of each station and then to check whether the total number of crossings of the implied solution is no more than c , which can be clearly done in polynomial time.

Let I be an instance of the fixed linear crossing number problem, consisting of a graph $G = (V, E)$, where $V = \{u_1, u_2, \dots, u_n\}$ and $E = \{e_1, e_2, \dots, e_m\}$, and a node ordering. Without loss of generality, we assume that $u_1 < u_2 < \dots < u_n$. We construct an instance I' of the MLCM-SE problem on a path as follows: The underlying network $G' = (V', E')$ is a path consisting of $n + 2$ nodes and $n + 1$ edges, where $V' = V \cup \{u_0, u_{n+1}\}$ and $E' = \{(u_{i-1}, u_i); 1 \leq i \leq n + 1\}$. The set of lines L is partitioned into two sets L^A and L^B :

- L^A consists of a sufficiently large number of lines (e.g., $2nm^2$ lines) connecting u_0 with u_{n+1} .
- L^B contains m lines l_1, l_2, \dots, l_m , one for each edge of G . Line l_i , which corresponds to edge e_i of G , has terminals at the end points of e_i .

Figure 5 illustrates the construction. First observe that all lines of L^A can be routed “in parallel” without any crossing among them (see Figure 5.b). Also observe that in an optimal solution none of the lines l_1, l_2, \dots, l_m crosses the lines of L^A , since that would contribute a very large number of crossings. Thus, in an optimal solution each line of L^B has both of its terminals either at top or at bottom station ends. This excludes any solution of I' where a line $l \in L^B$ has one of its terminals at a top station end and the other one at a bottom station end.

Assume now that there exists an optimal linear embedding of I with c crossings. We first route the lines of L^A without introducing any crossing among them. The remaining lines l_1, l_2, \dots, l_m will be routed either above or below the lines of L^A depending on the placement (i.e., either above or below the x -axis) of their corresponding edges in the embedding of I (refer to Figure 5.b). This implies a one-to-one correspondence between the crossings among pairs of edges of I and the crossings among pairs of lines of I' , as desired.

Consider now the case where we have determined an optimal solution of I' with c pairs of crossing lines. As already mentioned, lines l_1, l_2, \dots, l_m do not cross the lines of the set L^A , since a solution including such a crossing is not optimal. Therefore, each line of L^B entirely lies either above the lines of L^A or

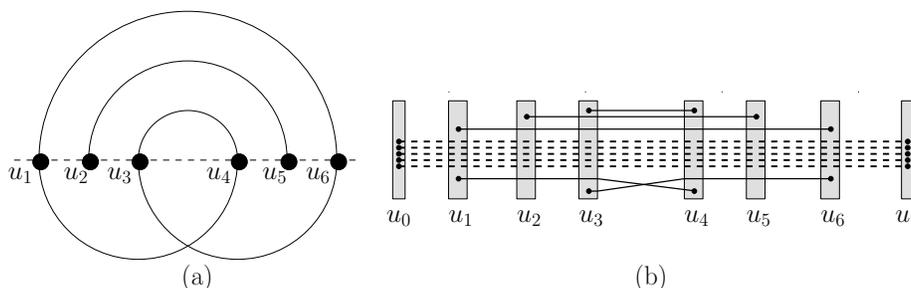


Figure 5: (a) A linear embedding, (b) an instance of the MLCM-SE problem.

below them. In the case where a line $l_i \in L^B$ lies above them, we draw edge e_i of graph G above the x -axis, otherwise below that. Again, it is easy to see that there exists an one-to-one correspondence between the crossings among pairs of edges of I and the crossings among pairs of lines of I' . Therefore, instance I has a linear embedding with c crossings among its edges. \square

3 The MLCM Problem under the k -Side Model

In this section, we study the MLCM problem under the k -side model, assuming that all line terminals are located at stations of degree one, which are referred to as *terminal stations* (see Figure 1). Stations of degree greater than one are referred to as *internal stations*. To simplify the description of our algorithm and to make the accompanying figures simpler, we restrict our presentation to the MLCM problem under the 4-side model, i.e., we assume that each station is represented as a rectangle and we permit tracks to all four sides of each station. Our algorithm for the case of k -side model is identical, since it is based on recursion over the edges of the underlying network. Recall that the lines can terminate at any track of their terminal stations, and the two sides of each station where each line enters or leaves the station are specified as part of the input. We further assume that an internal station always exists within the underlying network, otherwise the problem can be solved trivially.

The basic idea of our algorithm is to decompose the underlying network by removing an arbitrary edge out of the edges that connect two internal stations (and, consequently, appropriately partitioning the set of lines that traverse this edge), then recursively solve the subproblem and, finally, derive a solution of the initial problem by i) re-inserting the removed edge and ii) connecting the partitioned lines along the re-inserted edge. In the following subsections, we present the base of the recursion and the recursive step.

3.1 Base of recursion

The base of the recursion corresponds to the case of a graph G_B consisting of an internal station u and a number of terminal stations, say v_1, v_2, \dots, v_f , incident

to u , each of which has only one side with terminals (see Figure 6). To cope with this case, we first group all lines that have exactly the same terminals into a single line, which is referred to as *bundle*. The lines in a bundle will be drawn in a uniform fashion, i.e., occupying consecutive tracks at their common stations. In an optimal solution, a bundle can be safely replaced by its corresponding lines without affecting the optimality of the solution. In Figure 6c, lines belonging to the same bundle have been drawn with the same type of non-solid line. We refer to single lines as bundles, too, in order to maintain a uniform terminology (refer to the solid lines of Figure 6c). Then, the number of bundles of each terminal station is bounded by the degree of the internal station u .

In order to route the bundles along the edges of G_B , we make use of what we call *Euler tour numbering*. Let v be a terminal station of G_B . Then, the Euler tour numbering of the terminal stations v_1, v_2, \dots, v_f of G_B with respect to v is a function $\text{ETN}_v : \{v_1, v_2, \dots, v_f\} \rightarrow \{0, 1, \dots, f-1\}$. More precisely, we number all terminal stations of G_B according to the order of first appearance when moving clockwise along the external face of G_B starting from station v , which is assigned the value zero. Note that such a numbering is unique with respect to v and we refer to it as the Euler tour numbering starting from station v or simply as ETN_v . In Figure 6c, the number next to each line terminal at each terminal station v_i corresponds to the ETN_{v_i} of its destination, $i = 1, 2, \dots, 8$. Also, note that the computation of only one numbering is enough in order to derive the corresponding Euler tour numberings from any other terminal station v' of G_B , since $\text{ETN}_{v'}(w) = (\text{ETN}_v(w) - \text{ETN}_v(v')) \bmod f$.

Our approach works as follows. We first sort the bundles at each terminal station v based on the Euler tour numbering starting from v (i.e., ETN_v) of their destinations in ascending order and we place them so that they appear in counterclockwise order around the internal station u (see Figure 6a). We denote by $\text{BND}(v)$ the ordered set of bundles of each terminal station v . Then, we pass

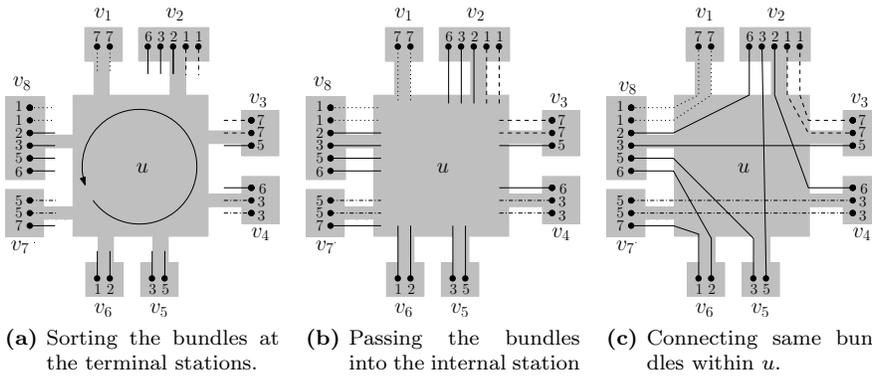


Figure 6: Illustration of the base of the recursion. The number next to each line terminal at each terminal station v_i corresponds to the ETN_{v_i} of its destination, $i = 1, 2, \dots, 8$.

these bundles from each terminal station to the internal station u along their common edge without introducing any crossings (see Figure 6b). This implies an ordering of the bundles at each side of the internal station u . To complete the routing procedure, it remains to connect equal bundles in the interior of the internal station u , which may imply crossings (see Figure 6c). Note that only station crossings that cannot be avoided are created in this manner since the underlying network is planar and since the Euler tour numbering implies that no unnecessary edge crossings occur. Therefore the optimality of the solution follows trivially.

3.2 Description of the recursive algorithm

Having fully specified the base of the recursion, we now proceed to describe our recursive algorithm in detail. First observe that, in the case where the input graph is not connected, we can separately solve the MLCM problem on each of the connected components of G and the lines induced by each of these components. Thus, in the rest of this section, we will assume that the input graph is connected. Let $e = (v, w)$ be an edge which connects two internal stations v and w of the underlying network. If no such edge exists, then the problem can be solved by employing the algorithm of the base of the recursion.

Let L_e be the set of lines that traverse e . Any line $l \in L_e$ originates from a terminal station, passes through a sequence of edges, then enters station v , traverses edge e , leaves station w and, finally, passes through a second sequence of edges until it terminates at another terminal station. Since each line consists of a sequence of edges, L_e can be written in the form $\{l \in L \mid l = \pi e \pi'\}$. We proceed by removing edge e from the underlying network and by inserting two new terminal stations t_e^v and t_e^w incident to the stations v and w , respectively (see the dark-gray colored stations of the right drawing of Figure 7). Let $G^* = (V \cup \{t_e^v, t_e^w\}, (E - \{e\}) \cup \{(v, t_e^v), (t_e^w, w)\})$ be the new underlying network obtained in this manner.

Since the edge e has been removed from the underlying network, the lines of L_e cannot traverse e anymore. So, we force them to terminate at t_e^v and t_e^w , as it is depicted in the right drawing of Figure 7. More precisely, let:

- $L_e^v = \{\pi (v, t_e^v) \mid \pi e \pi' \in L_e\}$
- $L_e^w = \{(t_e^w, w) \pi' \mid \pi e \pi' \in L_e\}$

Then, the new set of lines that is obtained after the removal of the edge e is $L^* = (L - L_e) \cup (L_e^v \cup L_e^w)$. We proceed by we recursively solving the MLCM problem on (G^*, L^*) .

The recursion will lead to a solution of (G^*, L^*) . Part of the solution consists of two ordered sets of bundles $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$ at each of the terminal stations t_e^v and t_e^w , respectively. Recall that, in the base of the recursion, all lines in a bundle have exactly the same terminals. In the recursive step, a bundle actually corresponds to a set of lines (with the same terminals) whose relative positions cannot be determined. In order to construct a solution of (G, L) , we

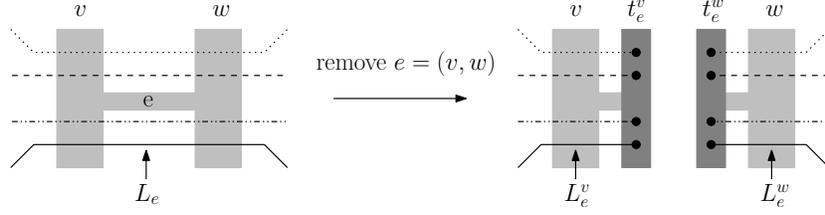


Figure 7: Illustration of the removal of an edge that connects two internal stations.

first have to restore the removed edge e and to remove the terminal stations t_e^v and t_e^w . The bundles $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$ of t_e^v and t_e^w have also to be connected appropriately along the edge e . Note that the order of the bundles of t_e^v and t_e^w is equal to those of v and w , because of the base of the recursion. Therefore, the removal of t_e^v and t_e^w will not produce unnecessary crossings.

We now proceed to describe the procedure of connecting the ordered bundle sets $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$ along edge e . We say that a *bundle is of size s* if it contains exactly s lines. We also say that two bundles are *equal* if they contain the same set of lines, i.e., the parts of the lines that each bundle contains correspond to the same set of lines. First, we connect all equal bundles. Let $b \in \text{BND}(t_e^v)$ and $b' \in \text{BND}(t_e^w)$ be two equal bundles. The connection of b and b' will result into a new bundle which i) contains the lines of b (or equivalently of b') and ii) its terminals are the terminals of b and b' that do not participate in the connection. Note that a bundle is specified as a set of lines and a pair of stations, that correspond to its terminals. When the connection of b and b' is completed, we remove both b and b' from $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$.

If both $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$ are empty, all bundles are connected. In the case where they still contain bundles, we determine the largest bundle, say b_{\max} of $\text{BND}(t_e^v) \cup \text{BND}(t_e^w)$. Without loss of generality, we assume that $b_{\max} \in \text{BND}(t_e^v)$ (see the left drawing of Figure 8). Since b_{\max} is the largest bundle among the bundles of $\text{BND}(t_e^v) \cup \text{BND}(t_e^w)$ and all equal bundles have been removed from both $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$, b_{\max} contains at least two lines that belong to different bundles of $\text{BND}(t_e^w)$. So, it can be split into smaller bundles, each of which contains a set of lines belonging to the same bundle in $\text{BND}(t_e^w)$ (see the right drawing of Figure 8). Also, the order of the new bundles in $\text{BND}(t_e^v)$ must follow the order of the corresponding bundles in $\text{BND}(t_e^w)$ in order to avoid unnecessary crossings (refer to the order of the bundles within the dotted rectangle of Figure 8). In particular, the information that a bundle was split is propagated to all stations that this bundle traverses, i.e., splitting a bundle is not a local procedure that takes place along a single edge but it requires greater effort. Note that no crossings among lines of b_{\max} occur, when b_{\max} is split. In addition, the crossings between lines of b_{\max} and bundles in $\text{BND}(t_e^w)$ cannot be avoided, which proves the correctness of our algorithm.

We repeat these two steps (i.e., connecting equal bundles and splitting the largest bundle) until both $\text{BND}(t_e^v)$ and $\text{BND}(t_e^w)$ are empty. Since we always split the largest bundle into smaller ones, this guarantees that our algorithm regard-

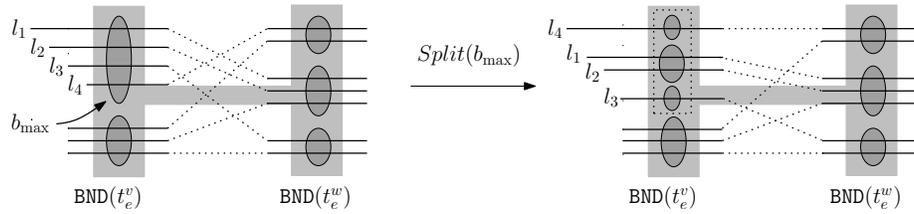


Figure 8: Splitting the largest bundle. The dotted lines just illustrate which connections have to be made. Note that no equal bundles exist.

ing the connection of the bundles along the edge e will eventually terminate. The basic steps of our algorithm are summarized in Algorithm 1.

Theorem 2 *Given a graph $G = (V, E)$ and a set L of lines on G that terminate at stations of degree 1, the metro-line crossing minimization problem under the k -side model can be solved in $O((|E| + |L|^2)|E|)$ time.*

Proof: The base steps of the recursion trivially take $O(|V| + |L|)$ time. The complexity of our algorithm is determined by step D of Algorithm 1, where the connection of bundles along a particular edge takes place. The previous steps of Algorithm 1 need $O((|V| + |E|)|E| + |V||L|)$ time in total. Since we always remove an edge that connects two internal stations, step D of Algorithm 1 will be recursively called at most $O(|E|)$ times.

In step $D.1$ of Algorithm 1, we have to connect equal bundles. To achieve this, we initially sort the lines of $\text{BND}(t_e^w)$ using counting sort [7] in $O(|L| + |L_e|)$ time, assuming that the lines are numbered from 1 to $|L|$, and we store them in an array, say B , such that the i -th numbered line occupies the i -th position of B . Then, all equal bundles can be connected by performing a single pass over the lines of each bundle of $\text{BND}(t_e^v)$. Note that, given a line l that belongs to a particular bundle of $\text{BND}(t_e^v)$, say b , we can determine in constant time to which bundle of $\text{BND}(t_e^w)$ it belongs by employing array B . So, in a total of $O(|b|)$ time, we decide whether b is equal to one of the bundles of $\text{BND}(t_e^w)$, which yields into an $O(|L_e|)$ total time for all bundles of $\text{BND}(t_e^v)$. Therefore, step $D.1$ of Algorithm 1 can be accomplished in $O(|L| + |L_e|)$ time.

Having connected all equal bundles, the largest bundle is then determined in $O(|m_e|)$ time, where $m_e = |\text{BND}(t_e^v) \cup \text{BND}(t_e^w)|$. In step $D.3$ of Algorithm 1, the largest bundle is split. Again, using counting sort [7] this can be accomplished in $O(|L| + |L_e|)$ time. The propagation of the splitting of the largest bundle in step $D.4$ of Algorithm 1 needs $O(|V||L_e|)$ time. The connection of the equal bundles and the splitting of the largest bundle will take place at most $O(|L_e|)$ times. Since $|m_e| \leq 2|L_e|$ and $|L_e| \leq |L|$, the total time needed for Algorithm 1 is $O((|E| + |V||L|^2)|E| + |V||L|)$.

Note that the above straight-forward analysis can be improved by a factor of $|V|$. This is accomplished by propagating the splitting of each bundle only to its endpoints (i.e., not to all stations that each individual bundle traverses). This

Algorithm 1: REC-DRAW(G, L)

input : An embedded, planar graph G and a set L of lines.**output** : A routing of the lines on G s.t. they cross each other as few times as possible.**require:** G contains at least one internal station.Rec-Draw($G=(V,E), L$)

{

{Recursion on the connected components of G .}**if** (G is not connected) **then****foreach** (connected component G^* of G) **do** $L(G^*) \leftarrow$ lines induced by G^* ;Rec-Draw($G^*, L(G^*)$);**return**;**if** (\nexists edge connecting internal stations) **then**

{Base of recursion.}

Apply the algorithm of the base of the recursion;

else

{Recursive step.}

 $e = (v, w) \leftarrow$ an edge connecting internal stations; $L_e \subseteq L \leftarrow$ set of lines that pass through e ;Let $L_e = \{l \in L \mid l = \pi e \pi'\}$;{Step A: Remove e , insert terminals t_e^v, t_e^w incident to v, w , resp.} $G^* \leftarrow (V \cup \{t_e^v, t_e^w\}, (E - \{e\}) \cup \{(v, t_e^v), (t_e^w, w)\})$

{Step B: Update the set of lines; see Figure 7}

 $L^* \leftarrow (L - L_e) \cup (L_e^v \cup L_e^w)$, where:• $L_e^v = \{\pi(v, t_e^v) \mid \pi e \pi' \in L_e\}$ • $L_e^w = \{(t_e^w, w)\pi' \mid \pi e \pi' \in L_e\}$

{Step C: Recursive call.}

Rec-Draw(G^*, L^*)

{Step D: Bundle connection.}

while ($\text{BND}(t_e^v) \neq \emptyset$) **do**{BND(t_e^v) and BND(t_e^w) obtained from the base of recursion}1. Connect equal bundles of BND(t_e^v) and BND(t_e^w);2. Remove connected bundles from BND(t_e^v) and BND(t_e^w);3. Split the largest bundle b_{\max} of BND(t_e^v) \cup BND(t_e^w);4. Propagate the splitting of b_{\max} to all stations that it traverses;

immediately implies that some stations of G may still contain bundles after the termination of the algorithm. So, we now need an extra post-processing step

to fix this problem. We use the fact that the terminals of G do not contain bundles, since they are always at the endpoints of each bundle, when it is split. This suggests that we can split—up to lines—all bundles at stations incident to the terminal stations. We continue in the same manner until all bundles are eventually split. Note that this extra step needs a total of $O(|E||L|)$ time and consequently does not affect the total complexity, which is now reduced to $O((|V| + |E| + |L|^2)|E| + |V||L|)$. Since G is connected, $|E| \geq |V| - 1$ and therefore Algorithm 1 needs $O((|E| + |L|^2)|E|)$ time, as desired. \square

4 The MLCM Problem under the 2-Side Model

In this section, we adopt the scenario of Section 3 under the 2-side model, i.e., we study the MLCM problem assuming that all line terminals are at stations of degree one, each station is represented as a rectangle, and tracks are permitted only to the left and to the right side of each station, i.e., one of the rectangle’s sides is devoted to “incoming” lines while the other one is devoted to “outgoing” lines (see Figure 9a). Since we do not permit a line to use the same side of a station to both enter and leave, all the lines are x -monotone. Note that, since the 2-side model is a restricted case of the 4-side model, Algorithm 1 can be applied in this case. However, our intention is to construct a more efficient algorithm.

Before we proceed with the description of our algorithm, we introduce some terminology. Since the lines are x -monotone, we refer to the leftmost and rightmost terminals of each line as its *origin* and *destination*, respectively. We also say that a line uses the left side of a station to *enter* it and the right side to *leave* it. Furthermore, we refer to the edges that are incident to the left (resp. right) side of each station u in the embedding of G as *incoming* (resp. *outgoing*) edges of station u (see Figure 9a). For each station u of G , the embedding of G also specifies an order of both the incoming and outgoing edges of u . We denote these orders by $E_{\text{in}}(u)$ and $E_{\text{out}}(u)$, respectively (see Figure 9a).

A key component of our algorithm is a numbering of the edges of G , i.e., a function $\text{EN} : E \rightarrow \{1, 2, \dots, |E|\}$. In order to obtain this numbering, we first construct a directed graph $G' = (V', E')$, as follows: For each edge $e \in E$ of G , we introduce a new vertex v_e in G' (refer to the little black disks in Figures 9b and 9c). Therefore, $|V'| = |E|$. For each pair of edges e and e' of G that are consecutive in that order in $E_{\text{in}}(u)$ or $E_{\text{out}}(u)$, where $u \in V$ is an internal station of G , we introduce an edge $(v_e, v_{e'})$ in G' (refer to the black-colored solid edges of Figure 9b). Finally, we introduce an edge connecting the vertex of G' associated with the last edge of $E_{\text{in}}(u)$ to the vertex of G' associated with the first edge of $E_{\text{out}}(u)$ (refer to the black-colored dashed edge of Figure 9b). Then, $|E'| = O(|E|)$. Using the embedding of G , we direct each edge of G' .

An illustration of the proposed construction is depicted in Figure 9c. Note that all edges of G' are either directed “downward” or “left-to-right” w.r.t. an internal station. This implies that there exist no cycles within the constructed graph. The desired numbering of the edges of G is then implied by performing a

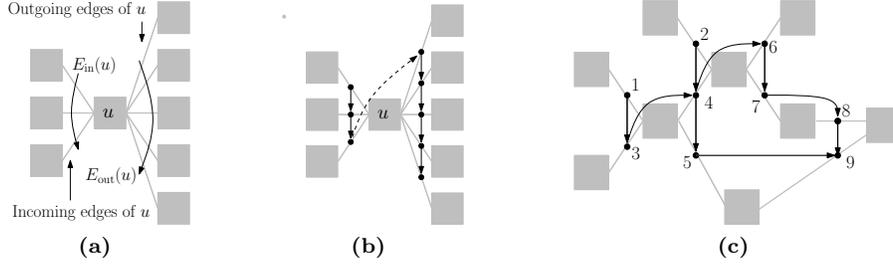


Figure 9: (a) An illustration of the incoming/outgoing edges of a station u . (b) An example of the construction of graph G' in the case where the underlying network consists of a single internal station u . (c) An edge numbering of the underlying network.

topological sorting on G' (see Figure 9c). From the construction of G' , it follows that the numbering obtained in this manner has the following properties:

- (i) The numbering of the incoming (resp. outgoing) edges of each internal station u follows the order $E_{in}(u)$ (resp. $E_{out}(u)$), i.e., an edge later in the order $E_{in}(u)$ (resp. $E_{out}(u)$) is assigned a greater number than an edge earlier in this order.
- (ii) The numbers assigned to the incoming edges of each internal station u are smaller than the corresponding numbers assigned to its outgoing edges.

Since each line is a sequence of edges, it can be equivalently expressed as a sequence of numbers based on the edge numbering $EN : E \rightarrow \{1, 2, \dots, |E|\}$. We refer to the sequence of numbers assigned to each line as its *numerical representation*. Note that the numerical representation of each line is sorted in ascending order because of the second property of the numbering and the x -monotonicity of the lines. We now proceed to consider two cases where a crossing between two lines cannot be avoided:

Unavoidable edge crossings: Let l and l' be two lines that share a common path of G . Let $a_1 \dots a_q \pi b_1 \dots b_r$ and $a'_1 \dots a'_g \pi b'_1 \dots b'_h$ be their numerical representations, respectively, where the subsequence π corresponds to the numerical representation of their common path. Then, it is easy to see that l and l' will inevitably cross if and only if $(a_q - a'_g) \cdot (b_1 - b'_1) < 0$. This case is depicted in Figure 10a. Note that the crossing of l and l' can be placed along any edge of their common path. This is because we aim to avoid unnecessary station crossings.

Unavoidable station crossings: Consider two lines l and l' that share only a single internal station u of the underlying network. We assume that u is incident to at least four edges, say e_v^1, e_v^2, e_v^3 and e_v^4 , where e_v^1 and e_v^2 are incoming edges of u , whereas e_v^3 and e_v^4 outgoing. We further assume that l enters u using e_v^1 and leaves u using e_v^4 . Similarly, l' enters u using e_v^2 and

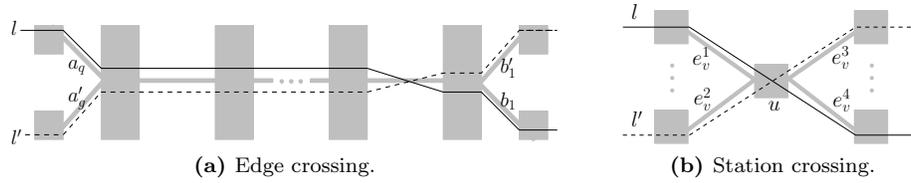


Figure 10: Crossings that cannot be avoided. Note that in Figure 10a, $a_q < a'_g < b'_1 < b_1$, whereas in Figure 10b, $\text{EN}(e_v^1) < \text{EN}(e_v^2) < \text{EN}(e_v^3) < \text{EN}(e_v^4)$.

leaves u using e_v^3 (see Figure 10b). Then, l and l' form a station crossing that cannot be avoided if and only if $(\text{EN}(e_v^1) - \text{EN}(e_v^2)) \cdot (\text{EN}(e_v^4) - \text{EN}(e_v^3)) < 0$. In this case, the crossing of l and l' can only be placed in the interior of station u .

Our intention is to construct a solution where only crossings that cannot be avoided are present. We will draw the lines of G incrementally by appropriately iterating over the stations of G and by extending the lines from previously considered stations to the current station. Assuming that the edges of G are directed from left to right in the embedding of G , we first topologically sort the stations of G . Note that, since all edges are directed from left to right, the graph does not contain cycles, and therefore a topological order exists. We consider the stations of G in their topological order. This ensures that, whenever we consider a station, its incoming lines have already been routed up to its left neighbors. Let u be the current station in the order. We distinguish the following two cases:

Case (a) : *The indegree of u is zero (i.e. u is a terminal station).*
 A station u with zero indegree corresponds to a station which only contains the origins of some lines. In this case, we simply sort these lines in ascending order lexicographically with respect to their numerical representations. This implies the desired ordering of the lines along the right side of station u .

Case (b) : *The indegree of u is greater than zero.*
 Let $e_u^1, e_u^2, \dots, e_u^d$ be the incoming edges of station u , where $d = \text{indegree}(u)$ and $e_u^i = (u_i, u)$, $i = 1, \dots, d$. Without loss of generality, we assume that $\text{EN}(e_u^i) < \text{EN}(e_u^j)$ if $i < j$. The lines that enter u from e_u^1 occupy the top-most tracks of the left side of station u . Then, the lines that enter u from e_u^2 occupy the next available tracks and so on.

Let L_u^i be the lines that enter u from edge e_u^i , $i = 1, 2, \dots, d$, ordered according to the order of the lines along the right side of station u_i . To specify the order of all lines along the left side of station u , it remains to describe how the lines of L_u^i are ordered when entering u , for each $i = 1, 2, \dots, d$. We stably sort in ascending order the lines of L_u^i based on the numbering of the edges that they use when leaving station u . In order

to perform this sorting we simply consider the number following $\text{EN}(e_u^i)$ in the numerical representation of each line.

Up to this point, we have specified the order of the lines along the left side of station u , say L_{in}^u . In order to complete the description of this case it remains to specify the order, say L_{out}^u , of these lines along the right side of u . Again, the desired order L_{out}^u is implied by stably sorting the lines of L_{in}^u based on the numbering of the edges that they use when they leave station u . Note that also in this case the sorting of the lines is performed by considering only the EN-number of the edges used by the lines when leave station u .

The basic steps of our algorithm are summarized in Algorithm 2. The lexicographical sortings taking place at terminal stations ensure that the lines that originate from each terminal station do not cross along their first common path. Furthermore, it is easy to see that the lines that enter each station from different edges do not cross each other when entering the station since they use non-conflicting tracks. To complete the proof of the correctness of Algorithm 2, it remains to prove that the stable sortings that are performed at each internal station ensure that only unavoidable station and edge crossings possibly occur. To see this, first consider two lines $l, l' \in L_u^i$ which enter a station u using the same edge e_u^i and use the same edge to leave station u . Since the sorting is stable, their relative position does not change when they enter u , which implies that they do not cross along the edge e_u^i . Thus, only unavoidable edge crossing are present and such crossings are always placed along the last edge of the common path of the crossing lines. Similarly, we can prove that none of the station crossings can be avoided. Now, we are ready to present the main theorem of this section.

Theorem 3 *Given a graph $G = (V, E)$ and a set L of lines on G that terminate at stations of degree 1, the metro-line crossing minimization problem under the 2-side model can be solved in $O(|V||E| + \sum_{l \in L} |l|) = O((|E| + |L|)|V|)$ time.*

Proof: Step A of Algorithm 2 needs $O(|V| + |E|)$ time in order to perform a topological sorting on G . In step B.1 of Algorithm 2, we have to construct graph G' and perform a topological sorting on it. This can be done in $O(|E|)$ time, since both the number of nodes and the number of edges of G' are bounded by $|E|$. Having computed the EN-number of each individual edge of the underlying network, the numerical representations of all lines in step B.2 of Algorithm 2 can be computed in $O(\sum_{l \in L} |l|)$ time.

Using radix sort [7], we can lexicographically sort all lines at each terminal station v of indegree zero in $O((|E| + |L_v|)|l_{\text{max}}^v|)$ total time, where L_v is the set of lines that originate at v and l_{max}^v is the longest line of L_v . Therefore, the sorting of all lines that fall into case (a) of step C needs a total of $O((|E| + |L|)|V|)$ time, since the length of the longest line of L is at most $|V|$.

In step C.1 of Algorithm 2, we stably sort the lines of each set L_u^i , $i = 1, 2, \dots, d$ based on the numbering of the edges that they use when leave u .

Algorithm 2: 2-SIDED ALGORITHM

input : An embedded, connected, planar graph G and a set of lines L .
output : A routing of the lines on G s.t. they cross each other as few times as possible.
require: Tracks are permitted to the left and the right side of each station.

{Step A: Topological Sort}
Perform a topological sorting on G , assuming that the edges of G are directed from left to right in the embedding of G ;

{Step B: Numerical representations.}
1. Determine the EN-number of each edge $e \in E$;
2. Rewrite all lines in numerical representation;

{Step C: Line Routing}
foreach station u in topological order **do**
 if ($\text{indegree}(u) = 0$) **then**
 {Case (a)}
 Sort the lines that originate from u lexicographically in ascending order;
 else
 {Case (b)}
 $d \leftarrow \text{indegree}(u)$;
 $e_u^1 \dots e_u^d \leftarrow$ incoming edges of u s.t. $e_u^i = (u_i, u)$ & $\text{EN}(e_u^i) < \text{EN}(e_u^j)$,
 for any $1 \leq i < j \leq d$;
 $L_u^i \leftarrow$ lines that enter u from edge e_u^i in the order in which they leave the right side of u_i ;
 {Step C.1: Find the order L_{in}^u of the lines along the left side of u }
 $L_{\text{in}}^u \leftarrow \emptyset$;
 for $i = 1$ **to** d **do**
 Stably sort the lines of L_u^i based on the numbering of the edges they use when leaving u and add them to the end of L_{in}^u ;
 {Step C.2: Find the order L_{out}^u of the lines along the right side of u }
 $L_{\text{out}}^u \leftarrow$ Stably sort the lines of L_{in}^u based on the numbering of the edges they use when leaving station u .

Using counting sort [7], this can be done in $O(|E| + |L_u|)$ total time, where L_u denotes the set of lines that traverse station u . Recall that counting sort is stable [7]. Similarly, step C.2 can be accomplished using counting sort and also needs $O(|E| + |L_u|)$ time. Summing over all internal stations, Algorithm 2 needs $O(|V||E| + \sum_{l \in L} |l|)$. \square

As already stated, we can employ Algorithm 2 to solve the MLCM-FixedSE problem. Our approach is as follows: For each station u of G , we introduce four new stations, say $u_{\text{left}}^{\text{top}}$, $u_{\text{left}}^{\text{bottom}}$, $u_{\text{right}}^{\text{top}}$ and $u_{\text{right}}^{\text{bottom}}$, adjacent to u . Station $u_{\text{left}}^{\text{top}}$

$(u_{\text{left}}^{\text{bottom}})$ is placed on top (below) and to the left of u in the embedding of G and contains all lines that originate at u 's top (bottom) station end. Similarly, station $u_{\text{right}}^{\text{top}}$ ($u_{\text{right}}^{\text{bottom}}$) is placed on top (below) and to the right of u in the embedding of G and contains all lines that are destined for u 's top (bottom) station end. In the case where some of the newly introduced stations contain no lines, we simply ignore their existence. So, instead of restricting each line to terminate at a top or at a bottom station end in its terminal stations, we equivalently assume that it terminates in one of the newly introduced stations. Then, Algorithm 2 can be used to solve the MLCM-FixedSE problem. The following theorem summarizes this result.

Theorem 4 *Given a graph $G = (V, E)$ and a set L of lines on G , the metro-line crossing minimization problem with fixed station ends under the 2-side model can be solved in $O(|V||E| + \sum_{l \in L} |l|)$ time.*

5 Conclusions

In this paper, we studied the MLCM problem under the k -side model for which we presented an $O((|E| + |L|^2)|E|)$ algorithm for the general case, and a more efficient algorithm for the special case of the 2-side model. Possible extensions would be to study the problem where the lines are not simple, and/or the underlying network is not planar. Our first approach seems to work even for these cases, although the time complexity is harder to analyze and cannot be estimated so easily. The focus of our work was on the case where all line terminals are located at specific stations of the underlying network. Allowing the line terminals anywhere within the underlying network would hinder the use of the proposed algorithms in both models. Therefore, it would be of particular interest to study the computational complexity of this problem. Another possible extension would be to try to derive approximation or fixed-parameter algorithms for the MLCM-SE problem, which was shown to be NP -complete.

References

- [1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:233–238, 1998.
- [2] J. Ahn and H. Freeman. AUTONAP - An expert system for automatic map name placement. In *Proc. International Symposium on Spatial Data Handling (SDH'84)*, pages 544–569, 1984.
- [3] E. Argyriou, M. A. Bekos, M. Kaufmann, and A. Symvonis. Two polynomial time algorithms for the metro-line crossing minimization problem. In I. G. Tollis and M. Patrignani, editors, *Proc. 16th International Symposium on Graph Drawing (GD'08)*, volume 5417 of *LNCS*, pages 336–347. Springer-Verlag, 2008.
- [4] M. Asquith, J. Gudmundsson, and D. Merrick. An ILP for the metro-line crossing problem. In J. Harland and P. Manyem, editors, *Fourteenth Computing: The Australasian Theory Symposium (CATS'08)*, volume 77 of *CRPIT*, pages 49–56. ACS, 2008.
- [5] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Line crossing minimization on metro maps. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. 15th International Symposium on Graph Drawing (GD'07)*, volume 4875 of *LNCS*, pages 231–242. Springer-Verlag, 2007.
- [6] M. Benkert, M. Nöllenburg, T. Uno, and A. Wolff. Minimizing intra-edge crossings in wiring diagrams and public transport maps. In M. Kaufmann and D. Wagner, editors, *Proc. 14th International Symposium on Graph Drawing (GD'06)*, volume 4372 of *LNCS*, pages 270–281. Springer-Verlag, 2006.
- [7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.
- [8] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
- [9] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annual ACM Symposium on Computational Geometry*, pages 281–288, 1991.
- [10] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
- [11] S.-H. Hong, D. Merrick, and H. Nascimento. The metro map layout problem. In N. Churcher and C. Churcher, editors, *Proc. Australasian Symposium on Information Visualisation (invis.au'04)*, volume 35 of *CRPIT*, pages 91–100. ACS, 2004.

- [12] K. G. Kakoulis and I. G. Tollis. On the edge label placement problem. In S. C. North, editor, *Proc. 4th International Symposium on Graph Drawing (GD'96)*, volume 1190 of *LNCS*, pages 241–256. Springer-Verlag, 1996.
- [13] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *LNCS*. Springer-Verlag, 2001.
- [14] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Trans. Comput.*, 39(1):124–127, 1990.
- [15] D. Merrick and J. Gudmundsson. Path simplification for metro map layout. In M. Kaufmann and D. Wagner, editors, *Proc. 14th International Symposium on Graph Drawing (GD'06)*, volume 4372 of *LNCS*, pages 258–269. Springer-Verlag, 2006.
- [16] M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In P. Healy and N. S. Nikolov, editors, *Proc. 13th International Symposium on Graph Drawing*, volume 3843 of *LNCS*, pages 321–333. Springer-Verlag, 2005.
- [17] S. H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.
- [18] J. M. Stott and P. Rodgers. Metro Map Layout Using Multicriteria Optimization. In *Proc. 8th International IEEE Conference on Information Visualisation (IV'2004)*, pages 355–362. IEEE Computer Society, 2004.
- [19] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.
- [20] A. Wolff and T. Strijk. The Map-Labeling Bibliography, maintained since 1996. On line: <http://i11www.ira.uka.de/map-labeling/bibliography>.
- [21] S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.
- [22] S. Zoraster. Practical results using simulated annealing for point feature label placement. *Cartography and GIS*, 24(4):228–238, 1997.