Two Polynomial Time Algorithms for the Metro-Line Crossing Minimization Problem

E. Argyriou[1], M. A. Bekos[1], M. Kaufmann[2] and A. Symvonis[1]

[1] School of Applied Mathematical & Physical Sciences,
National Technical University of Athens, Greece
{fargyriou,mikebekos,symvonis}@math.ntua.gr
[2] University of Tübingen, Institute for Informatics, Germany
mk@informatik.uni-tuebingen.de

Abstract. The *metro-line crossing minimization (MLCM)* problem was recently introduced in [5] as a response to the problem of drawing metro maps or public transportation networks, in general. According to this problem, we are given a planar, embedded graph $G = (V, E)$ and a set L of simple paths on G, called *lines*. The main task is to place the lines on the embedding of G, so that the number of crossings among pairs of lines is minimized.
Our main contribution is two polynomial time algorithms. The first solves the general case of the MLCM problem, where the lines that traverse a particular vertex of G are allowed to use any side of it to either "enter" or "exit", assuming that the endpoints of the lines are located at vertices of degree one. The second one –which is more efficient in terms of time complexity– solves the restricted case, where only the left and the right side of each vertex can be used.
To the best of our knowledge, this is the first time where the general case of the MLCM problem is solved. Previous work in the graph drawing literature was devoted to the restricted case of the MLCM problem under the additional assumption that the endpoints of the lines are either the topmost or the bottommost in their corresponding vertices, i.e., they are either on top or below the lines that pass through the vertex. Even for this case, we improve a known result of Asquith et al. [3] from $O(|E|^{5/2}|L|^3)$ to $O(|V|(|E| + |L|))$.

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 over the map, 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 more as electrical schematics, on which all stations are almost equally spaced, rather than true maps (see Fig.7 in Appendix A).

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 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 has to be performed over the most important features.

In the graph drawing and computational geometry literature, the focus so far has been nearly exclusively on the first and the third step. Closely related to the first step are the works of Hong et al. [8], Merrick and Gudmundsson [11], Nöllenburg and Wolff [12] and Stott and Rodgers [14]. The map labeling problem has also attracted the interest of several researchers. Even if the majority of map labeling problems are shown to be NP-complete [7, 9, 10, 13], several approaches have been suggested, among them expert systems [2], approximation algorithms [1, 7, 13, 15], zero-one integer programming [17], simulated annealing [18]. An extensive bibliography on map labeling is maintained by Strijk and Wolff [16].

The intermediate problem of adding the line set into the underlying network was recently introduced by Benkert et al. [5], followed by [4]. Since crossings within a visualization are often considered as the main source of confusion, the main goal is to draw the lines, so that they cross each other as few times as possible. This problem is referred to as the *metro-line crossing minimization problem (MLCM)*.

1.1 Problem Definition

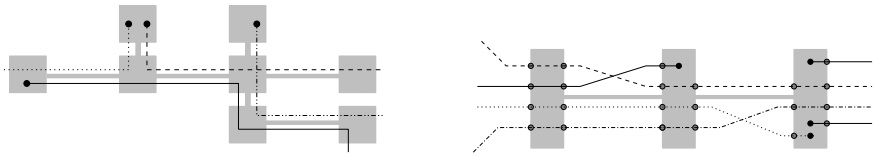
The input of the metro-line crossing minimization problem consists of a connected, embedded, planar graph $G = (V, E)$ and a set $L = \{l_1, l_2 \dots l_k\}$ of simple paths on G, called *lines*. We will refer to G as the underlying network and to the nodes of G as *stations*. We also refer to the endpoints of each line as its *terminals*. In this paper, we study the case where all line terminals are located at stations of degree one, which are referred to as *terminal stations*, see again Fig.7. Stations of degree greater than one are referred to as *internal stations*. 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 "exit" the station are also specified as part of the input. Motivated by the fact that a line cannot make a 180^o turn within a station, we do not permit a line to use the same side of a station to both "enter" and "exit".

The output of the MLCM problem should specify 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_i 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_i. Equivalently, we say that l_i *terminates* or *has terminals at* v_0 and v_d. By $|l_i|$ we denote the length of line l_i.

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 "exits" u). These points are referred to as *tracks* (see the dark-gray colored bullets on the boundary of each station in Fig.1b). In general, we may permit tracks to all sides of each station, i.e., a line that traverses a station may use any side of it to either "enter" or "exit" (see Fig.1a). In the case where the stations are represented as rectangles, this model is referred to as 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 k-*side* model. A more restricted model, referred to as 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 Fig.1b).

A particularly interesting case that arises under the 2-side model is the one where the lines that terminate at a station occupy its topmost and bottommost tracks, in



(a) 4-side model - Station crossings. **(b)** 2-side model - Edge crossings.

Fig. 1: The underlying network is the gray colored graph. We have used different types of lines to denote different lines.

the following referred to as *top* and *bottom station ends*, respectively (see Fig.1b). This is to emphasize that the line terminates at that station. The variant of the MLCM problem that fulfills this restriction is referred to as the *metro-line crossing minimization problem with station ends* (*MLCM-SE*). If additionally, the information whether a line terminates at a top or at a bottom station end in its terminal station is specified as part of the input, the corresponding problem is referred to as *metro-line crossing minimization problem with fixed station ends* (*MLCM-FixedSE*).

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 Fig.1b). We call this an *edge crossing*. As opposed to an edge crossing, a *station crossing* occurs inside a station. For aesthetic reasons, we want to avoid station crossings whenever this is possible (e.g. in the case of 4-side model this is not always feasible; see Fig.1a).

1.2 Previous Work and our Results

The first results on the MLCM problem were presented by Benkert et al. in [5], who devised a dynamic-programming algorithm that runs in $O(|L|^2)$ time for the restricted case where the crossings are minimized along a single edge of G. Bekos et al. [4] proved that the MLCM-SE problem is NP-complete even in the case where the underlying network is a path. They also proved that the MLCM-FixedSE problem can be solved in $O(|V| + \log d \sum_{i=1}^{|L|} |l_i|)$, in the case where the underlying network is a tree of degree d. Extending the work of Bekos et al., Asquith et al. [3] proved that the MLCM-FixedSE problem is also solvable in polynomial time in the case where the underlying network is 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.

This paper is structured as follows: In Section 2, we present a polynomial time algorithm, which 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 general model. In Section 3, we present a faster algorithm for the special case of 2-side restriction. The time complexity of the proposed algorithm is $O(|V||E| + \sum_{i=1}^{|L|} |l_i|)$. It can also be employed to solve the MLCM-FixedSE problem, which drastically improves the running time of the algorithm of Asquith et al. [3] from $O(|E|^{5/2}|L|^3)$ to $O(|V||E| + |V||L|)$. Outlines of the basic steps of our algorithms are also provided in Appendix B in order to facilitate the reviewers. We conclude in Section 4 with open problems and future work.

2 The MLCM Problem under the k-side Model

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 all line terminals are located at stations of degree one, the lines can terminate at any track of their terminal stations, and, finally, the sides of each station that each line may use to either "enter" or "exit" 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.

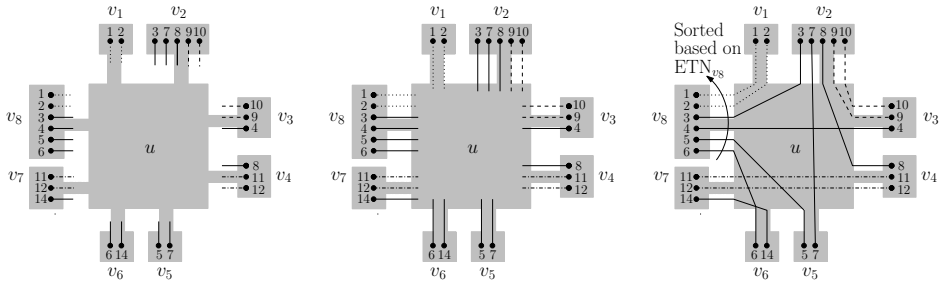
The basic idea of our algorithm is to decompose the underlying network by removing an arbitrary edge out of the edges that connect two internal stations (and consequently partitioning the set of lines that traverse this edge appropriately), then recursively solve the subproblem and, finally, derive a 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 sections, we present the base of the recursion and the recursive step.

2.1 Base of recursion

The base of the recursion corresponds to the case of a graph G_B consisting of a "central station" u containing no terminals and a particular number of terminal stations incident to u, let $v_1, v_2 \dots v_d$ (see Fig.2c). 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 notion of bundles corresponds to the fact that lines with same terminals are drawn in a uniform fashion, i.e., occupying consecutive tracks at their common stations. So, in an optimal solution once a bundle is drawn, it can be safely replaced by its corresponding lines without affecting the optimality of the solution. In Fig.2c, lines belonging to the same bundle have been drawn with the same type of non-solid line. Note that single lines are also treated as bundles in order to maintain a uniform terminology (refer to the solid lines of Fig.2c). Then, the number of bundles of each terminal station is bounded by the degree of the "central station" u.

In order to route the bundles along the edges of G_B, we will make use of the *Euler tour numbering* that was proposed by Bekos et al. in [4]. Let v be a terminal station of G_B. Then, the Euler tour numbering of the terminal stations $v_1, v_2, \dots, v_d$ of G_B with respect to v is a function $\texttt{ETN}_v : \{v_1, v_2, \dots, v_d\} \rightarrow \{0, 1, \dots, d-1\}$. More precisely, given a terminal station v of G_B, 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 zero value. 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 $\texttt{ETN}_v$. Also, note that the computation of only one numbering is enough in order to compute the corresponding Euler tour numberings from any other terminal station of G_B, since $\texttt{ETN}_{v'}(w) = (\texttt{ETN}_v(w) - \texttt{ETN}_v(v')) \mod d$.

Our approach is outlined as follows: We first sort in ascending order the bundles at each terminal station v based on the Euler tour numbering $\texttt{ETN}_v$ of their destinations (see Fig.2a). This implies the desired ordering of the bundles along the side of each



(a) Sorting the bundles at the terminal stations. **(b)** Passing the bundles into the central station **(c)** Connecting same bundles within u.

Fig. 2: Illustration of the base of the recursion. The numbering of the lines is arbitrary

terminal station that is incident to the "central station" u. We will denote by $\texttt{BND}(v)$ the ordered set of bundles of each terminal station v. Then, we pass these bundles from each terminal station to the "central station" u along their common edge without introducing any crossings (see Fig.2b). This will also imply an ordering of the bundles at each side of the "central station" u. To complete the routing procedure, it remains to connect same bundles in the interior of the "central station" u, which may imply crossings (see Fig.2c). Our approach is also outlined in Alg.1 (see Appendix B). Note that only necessary station crossings are created in this manner, since the underlying network is planar and from the Euler tour numbering it follows that no edge crossings will eventually occur. Therefore the optimality of the solution follows trivially.

2.2 Description of the recursive algorithm

Having fully specified the base of the recursion, we now proceed to describe our recursive algorithm in detail. 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 Alg.1 (base of the recursion).

Let L_e be the set of lines that traverse e. Any line $l_i^e \in L_e$ originates from a terminal station, passes through a sequence of edges, then enters station v, traverses edge e, exits station w and, finally, passes through a second sequence of edges until it terminates at another terminal station. Let $p : E \times L \to \mathbb{N}$ be a function, such that $p(e, l)$ denotes the position of edge e along the line l. Formally, $L_e = \{l_{e,1}, l_{e,2}, \ldots, l_{e,|L_e|}\}$, where $l_{e,i}$ denotes the i-th line of L_e. Since each line of L_e consists of a sequence of edges, L_e can be written in the form $\{l_{e,i} = e_{e,i}^1\ e_{e,i}^2\ \ldots\ e_{e,i}^{k-1}\ e\ e_{e,i}^{k+1}\ \ldots\ e_{e,i}^{|l_{e,i}|};\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}$. 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 Fig.3). 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 it any more. So, we force them to terminate at t_e^v and t_e^w, as it is depicted in right drawing of Fig.3. This is done by splitting the set L_e into two new sets L_e^v and L_e^w (see Fig.3), which are formally defined as follows:

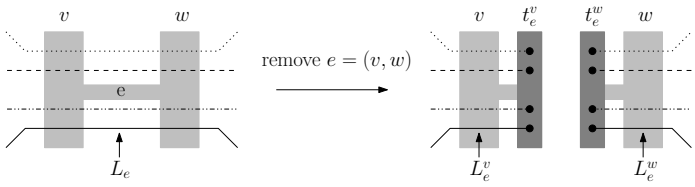


Fig. 3: Illustration of the removal of an edge that connects two internal stations.

- $L_e^v = \{e_{e,i}^1\ e_{e,i}^2\ \ldots\ e_{e,i}^{k-1}\ (v, t_e^v);\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}$
- $L_e^w = \{(t_e^w, w)\ e_{e,i}^{k+1}\ \ldots\ e_{e,i}^{|l_{e,i}|};\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}\}$

The new set of lines that it is obtained after the removal of the edge e is $L^* = (L - L_e) \cup (L_e^v \cup L_e^w)$. Also, observe that the removal of the edge e from the underlying network may disconnect it. So, we distinguish two cases. In the case where G^* is connected, we recursively solve the MLCM problem on (G^*, L^*). Otherwise, since G^* was obtained from G by the removal of a single edge, it has exactly two connected components, say G_1^* and G_2^*. Let $L(G_i^*)$ denotes the lines of L^* induced by G_i^*. In this case, we recursively solve the MLCM problem on $(G_1^*, L(G_1^*))$ and $(G_2^*, L(G_2^*))$.

The recursion will lead to a solution of (G^*, L^*). Part of the solution consists of two ordered sets of bundles $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$ at each of the terminal stations t_e^v and t_e^w, respectively. Recall that in the base of the recursion those lines are in the bundles that have exactly the same terminals. In the recursive step, a bundle actually corresponds to a set of lines whose relative positions cannot be determined. In order to construct a solution of (G, L), first we have to restore the removed edge e and remove the terminal stations t_e^v and t_e^w. The bundles $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$ 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 $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$ along the edge e. We say that a *bundle is of size* k iff it contains exactly k lines. We also say that two bundles are *equal* iff 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 $b \in \texttt{BND}(t_e^v)$ and $b' \in \texttt{BND}(t_e^v)$ 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 $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$.

If both $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$ are empty, all bundles are connected. In the case where they still contain bundles, we determine the largest in size bundle, say b_{max}, of $\texttt{BND}(t_e^v) \cup \texttt{BND}(t_e^w)$. Without loss of generality, we assume that $b_{max} \in \texttt{BND}(t_e^v)$ (see the left drawing of Fig.4). Since b_{max} is the largest bundle among the bundles of $\texttt{BND}(t_e^v) \cup \texttt{BND}(t_e^w)$ and all equal bundles have been removed from both $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$, b_{max} contains at least two lines that belong to different bundles of $\texttt{BND}(t_e^w)$. So, it can be split into smaller bundles, each of which contains a set of lines belonging to the same bundle in $\texttt{BND}(t_e^w)$ (see the right drawing of Fig.4). Also, the order of the new bundles in $\texttt{BND}(t_e^v)$ should follow the order of their corresponding bundles

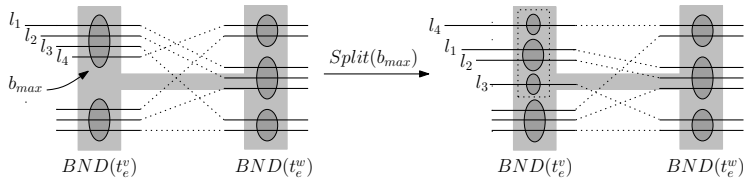


Fig. 4: Splitting the largest bundle. The dotted lines just illustrate which connections have to be made. Note that no equal bundles exist.

in $\texttt{BND}(t_e^w)$ in order to avoid unnecessary crossings (refer to the order of the bundles within the dotted rectangle of Fig.4). In particular, the information that a bundle was split should be 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 the crossings between lines of b_{max} and bundles in $\texttt{BND}(t_e^w)$ cannot be avoided. In addition, no crossings among lines of b_{max} occur.

We repeat these two steps (i.e. connection of equal bundles and splitting the largest bundle) until both $\texttt{BND}(t_e^v)$ and $\texttt{BND}(t_e^w)$ are empty. Since we always split the largest bundle into smaller ones, this guarantees that our algorithm regarding the connection of the bundles along the edge e will eventually terminate. The basic steps of our algorithm are also summarized in Alg.2 (see Appendix B).

Theorem 1. *Given a graph $G = (V, E)$ and a set of lines L on G that terminate at stations of degree* 1*, the metro-line crossing minimization problem under the* 4*-side model can be solved in $O((|E| + |L|^2)|E|)$ time.*

Proof. The base steps of the recursion (refer to Alg.1) trivially take $O(|V|+\sum_{i=1}^{|L|}|l_i|)$, or simpler, $O(|V||L|)$ total time. In particular, the complexity of our algorithm is determined by Step D of Alg.2, where the connection of bundles along a particular edge takes place. The previous steps of Alg.2 need $O((|V| + |E|)|E| + |V||L|)$ time in total. Since we always remove an edge that connects two internal stations, Step D of Alg.2 will be recursively called at most $O(|E|)$ times.

In Step$D$.1 of Alg.2, we have to connect equal bundles. To achieve this, we initially sort the lines of $\texttt{BND}(t_e^w)$ using counting sort [6] 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 are connected by performing a single pass over the lines of each bundle of $\texttt{BND}(t_e^v)$. Note that given a line l that belongs to a particular bundle, say b, of $\texttt{BND}(t_e^v)$, we can determine in constant time to which bundle of $\texttt{BND}(t_e^w)$ it belongs by employing array B. So, in a total of $O(|b|)$ we decide whether b is equal to one of the bundles of $\texttt{BND}(t_e^w)$, which yields into a total of $O(|L_e|)$ time for all bundles of $\texttt{BND}(t_e^v)$. Therefore, Step$D$.1 of Alg.2 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 $m_e = \texttt{BND}(t_e^v) \cup \texttt{BND}(t_e^w)$. In Step D.3 of Alg.2, the largest bundle is split. Again, using counting sort this can be accomplished in $O(|L| + |L_e|)$ time. The propagation of the splitting of the largest bundle in Step D.4 of Alg.2 needs $O(|V||L_e|)$ time. The connection of the equal bundles and the splitting of the largest bundle can be done at most $O(|L_e|)$ times. Since $|m_e| \leq 2|L_e|$ and $|L_e| \leq |L|$, the total time needed for Alg.2 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 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 Alg. 2 needs $O((|E|+|L|^2)|E|)$ time, as desired. □

As already stated, exactly the same algorithm can be used to solve the k-side model. This is summarized in the following theorem.

Theorem 2. *Given a graph $G = (V, E)$ and a set of lines L 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.*

3 The MLCM Problem under the 2-side Model

In this section, we adopt the scenario of Section 2 under the 2-side model, i.e., we study the MLCM problem assuming that each station is represented as a rectangle and we permit tracks to the left and the right side of each station[3], i.e., one of the rectangle's sides is devoted to "incoming" edges/lines while the other is devoted to "outgoing" edges/lines (see Fig.5a). This assumption combined with the fact that we do not permit a line to use the same side of a station to both "enter" and "exit" implies that all lines should be x-monotone.

Before we proceed with the description of our algorithm, we first introduce some necessary terminology, which is used in the remainder of this section. Since the lines are x-monotone, we refer to the leftmost (resp. rightmost) terminal of each line as its *origin* (resp. *destination*). We also say that a line uses the left side of a station to *enter* it and the right side to *exit* 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 Fig.5a). 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 $E_{in}(u)$ and $E_{out}(u)$, respectively (see Fig.5a).

A key component of our algorithm is a numbering of the edges of G, i.e., a function $\texttt{EN} : E \rightarrow \{1, 2, \ldots, |E|\}$. 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 black-colored bullets of Figures 5b and 5c). Therefore, $|V'| = |E|$. Also, for each pair of edges e_i and e_{i+1} of G that are consecutive in that order in $E_{in}(u)$ or $E_{out}(u)$, where $u \in V$ is an internal station of G, we introduce an edge $(v_{e_i}, v_{e_{i+1}})$ in G' (refer to the black-colored solid edges of Fig.5b). Finally, we introduce an edge connecting the vertex of G' associated with the last edge of $E_{out}(u)$

[3] Note that since the 2-side model is a restricted case of the 4-side model, Alg.2 can be applied in this case, too. Our intention is to construct a more efficient algorithm.

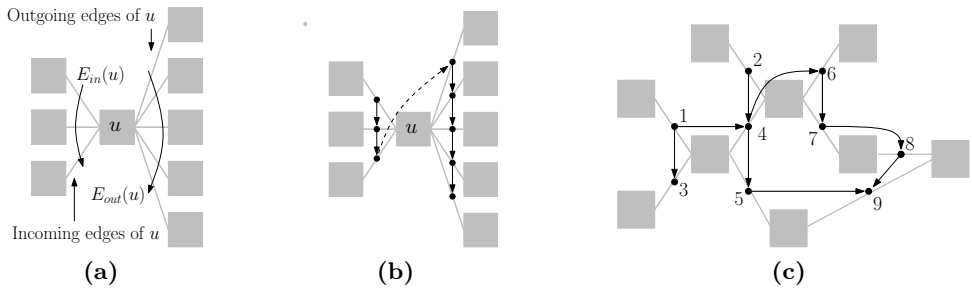


Fig. 5: (a) An illustration of the incoming/outgoing edges of a station u. (b) An example of the construction of graph G' in the case where the underlying network consists of a single internal station u. (c) An edge numbering of the underlying network.

to the vertex of G' associated with the first edge of $E_{in}(u)$ (refer to the black-colored dashed edge of Fig.5b). Then, $|E'| = O(|E|)$.

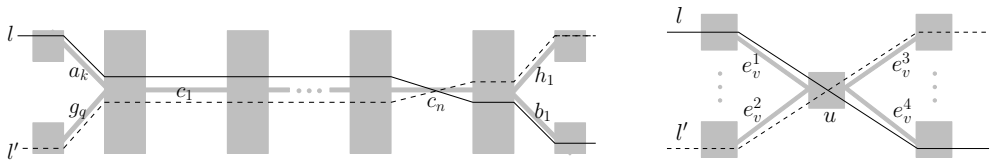
An illustration of the proposed construction is depicted in Fig.5c. Note that all edges of G' are either directed "downward" or "left-to-right" with respect to an internal station. This implies that there exist no cycles within the constructed graph (no "right-to-left" edges exist to form cycles). The desired numbering of the edges of G is then implied by performing a topological sorting on G' (see Fig.5c). From the construction of G' 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 $\texttt{EN} : E \to \{1, 2, \ldots, |E|\}$. 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 also $a_1a_2 \ldots a_kc_1c_2 \ldots c_mb_1b_2 \ldots b_n$ and $g_1g_2 \ldots g_qc_1c_2 \ldots c_mh_1h_2 \ldots h_r$ be their numerical representations, respectively, where the subsequence $c_1c_2 \ldots c_m$ corresponds to their common path. Then, it is easy to see that l and l' will inevitably cross iff $(a_k - g_q) \times (b_1 - h_1) < 0$. This case is depicted in Fig.6a. 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 exits u using e_v^4. Similarly, l' enters u using e_v^2 and exits u using e_v^3 (see Fig.6b). Then,



(a) Edge crossing. **(b)** Station crossing.

Fig. 6: Crossings that cannot be avoided. Note that in Fig.6a, $a_k < g_q < h_1 < b_1$, whereas in Fig.6b, $\texttt{EN}(e_v^1) < \texttt{EN}(e_v^2) < \texttt{EN}(e_v^3) < \texttt{EN}(e_v^4)$.

l and l' form a station crossing which cannot be avoided iff $(\texttt{EN}(e_v^1) - \texttt{EN}(e_v^2)) \times (\texttt{EN}(e_v^4) - \texttt{EN}(e_v^3)) < 0$. 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 iterated stations to the next station. Assuming that the edges of G are directed from left to right in the embedding of G, we first perform a topological sorting of the stations of G. Note that since all edges are directed from left to right, the graph does not contain cycles (no right to left edges exist to form cycles) and therefore a topological order exists. We consider the stations of G in their topological order. This ensures that whenever we consider the next station, its incoming lines have already been routed up to its left neighbors. Let u be the next station in the order. We distinguish the following cases:

Case (a) : $indegree(u) = 0$ *(i.e. terminal station).*
A station u with $indegree(u) = 0$ corresponds to a station which only contains the origins of some lines. In this case, we simply sort in ascending order these lines lexicographically with respect to their numerical representations. This implies the desired ordering of the lines along the right side of station u. It also ensures that these lines do not cross along their first common path.

Case (b) : $indegree(u) > 0$.
Let $e_u^1, e_u^2, \ldots, e_u^k$ be the incoming edges of station u, where $k = indegree(u)$ and $e_u^i = (u_i, u)$, $i = 1, \ldots, k$. W.l.o.g. we assume that $\texttt{EN}(e_u^i) < \texttt{EN}(e_u^j)$, $\forall i < j$. The lines that enter u from e_u^1 will occupy the topmost tracks of the left side of station u. Then, the lines that enter u from e_u^2 will occupy the next available tracks and so on. This ensures that the lines that enter u from different edges will not cross with each other, when entering u.
Let L_u^i be the lines that enter u from edge e_u^i, $i = 1, 2, \ldots, k$, ordered according to the order of the lines along the right side of station u_i. In order 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 $i = 1, 2, \ldots, k$. We stably sort in ascending order the lines of L_u^i based on the numbering of the edges that they use when exit station u. Note that in order to perform this sorting we only consider the number following $\texttt{EN}(e_u^i)$ in the numerical representation of each line. Also, the stable sorting ensures that only unavoidable edge crossings will occur along e_u^i. To see this consider two lines $l, l' \in L_u^i$ which use the same edge to exit station u. Since the sorting is stable their relative position will not change when they enter u, which implies that they will not cross along the edge e_u^i.

Up to this point, we have specified the order of the lines along the left side of station u, say L^u_{in}. In order to complete the description of this case it remains to specify the order, say L^u_{out}, of these lines along the right side of u. Again, the desired order L^u_{out} is implied by stably sorting the lines of L^u_{in} based on the numbering of the edges that they use when they exit 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 exit station u. Again, the stable sorting ensures that only unavoidable station crossings will occur in the interior of station u.

The basic steps of our algorithm are summarized in Alg.3 (see Appendix B). Note that the stable sortings that are performed at each terminal station ensure that only unavoidable station and edge crossings eventually occur. Also, an unavoidable edge crossing between two lines is always placed along the last edge of their common path.

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

Proof. Step A of Alg.3 needs $O(|V|+|E|)$ time in order to perform a topological sorting on G. In Step $B.1$ of Alg.3, 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 Alg.3 can be computed in $O(\sum_{i=1}^{|L|} |l_i|)$ time.

Using radix sort [6], we can lexicographically sort all lines at each terminal station v of indegree zero in $O((|E| + |L_v|)|l^v_{max}|)$ total time, where L_v is the set of lines that originate at v and l^v_{max} 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 Alg.3, we have to –stably– sort the lines of each set L^i_u, $i = 1, 2, \ldots, k$ based on the numbering of the edges that they use when exit u. Using counting sort, 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. Similarly, Step $C.2$ can be accomplished using counting sort and also needs $O(|E| + |L_u|)$ time. Summing over all internal stations, Alg.3 needs $O(|V||E| + \sum_{i=1}^{|L|} |l_i|)$. □

As already stated, we can employ Alg.3 to solve the MLCM-FixedSE problem. Our approach is as follows: For each station u of G, we introduce four new stations, say u^t_l, u^b_l, u^t_r and u^b_r, adjacent to u. Station u^t_l (u^b_l) 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^t_r (u^b_r) 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 to one of the newly introduced stations. Then, Alg.3 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 of lines L on G, the metro-line crossing minimization problem with fixed station ends under the 2-side model can be solved in $O(|V||E| + \sum_{i=1}^{|L|} |l_i|)$ time.*

4 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, and a more efficient algorithm for the special case of 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.

References

1. P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. In *Proc. 9th Canad. Conf. Comput. Geom.*, pages 233–238, 1997.
2. J. Ahn and H. Freeman. AUTONAP an expert system for automatic map name placement. In *Proc. Int. Symposium on Spatial Data Handling*, pages 544–569, 1984.
3. M. Asquith, J. Gudmundsson, and D. Merrick. An ILP for the metro-line crossing problem. In *Proc. 14th Computing: The Australasian Theory Symp.*, pages 49–56, 2008.
4. M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Line crossing minimization on metro maps. In *Proc. 15th Int. Symp. on Graph Drawing*, pages 231–242, 2008.
5. M. Benkert, M. Nöllenburg, T. Uno, and A. Wolff. Minimizing intra-edge crossings in wiring diagrams and public transport maps. In *Proc. 14th Int. Symp. on Graph Drawing*, pages 270–281, 2006.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.
7. M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.
8. S.-H. Hong, D. Merrick, and H. Nascimento. The metro map layout problem. In *Proc. Australasian Symp. on Information Visualisation*, pages 91–100, 2004.
9. C. Iturriaga and A. Lubiw. NP-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo, Canada, 1997.
10. K. G. Kakoulis and I. G. Tollis. On the edge label placement problem. In *Proc. of the Symposium on Graph Drawing '96*, pages 241–256, 1997.
11. D. Merrick and J. Gudmundsson. Path simplification for metro map layout. In *Proc. 14th Int. Symp. on Graph Drawing*, pages 258–269, 2006.
12. M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In *Proc. 13th Int. Symp. on Graph Drawing*, pages 321–333, 2006.
13. S. H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.
14. J. M. Stott and P. Rodgers. Metro map layout using multicriteria optimization. In *Proc. 8th Int. Conference on Information Visualisation*, pages 355–362. IEEE, 2004.
15. M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Conputational Geomentry: Theory and applications*, 13:21–47, 1999.
16. A. Wolff and T. Strijk. The Map-Labeling Bibliography, maintained since 1996. On line: http://i11www.ira.uka.de/map-labeling/bibliography.
17. S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.
18. S. Zoraster. Practical results using simulated annealing for point feature label placement. *Cartography and GIS*, 24(4):228–238, 1997.

A A Sample Metro Map

Fig. 7: An illustration of the metro map of Washington DC.

B Algorithms' Outlines

Algorithm 1: DRAW-BASE

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 : G consists of a "central station" u containing no terminals and a particular number of terminal stations incident to u, say $v_1, v_2 \dots v_d$.
{*Step A:* Create bundles}
Group all lines that have exactly the same terminals into a single bundle;
{*Step B:* Compute an Euler Tour Numbering}
Compute an Euler tour numbering of the terminal stations of G starting from v_1;
{*Step C:* Line Routing}
foreach terminal station v **do**
1. Sort in ascending order the bundles of v based on $\texttt{ETN}_v$;
2. Store the sorted set of bundles in $\texttt{BND}(v)$;
3. Pass the bundles from v to u along their common edge without crossings;

Connect same bundles in the interior of u.

Algorithm 2: REC-DRAW(G,L)

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 : At least one internal station exists within the initial underlying network.

`Rec-Draw(`G=(V,E), L`)`
{
  if ($\nexists$ edge connecting internal stations) **then** `/* Base of recursion */`
    Apply `Draw-Base` on (G,L);
  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 $L_e = \{l_{e,i} = e^1_{e,i}\ \ldots\ e^{k-1}_{e,i}\ e\ e^{k+1}_{e,i}\ \ldots\ e^{|l_{e,i}|}_{e,i};\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}$;
    {*Step A:* Remove e and insert two terminals t^v_e, t^w_e incident to v and w.}
    $G^* \leftarrow (V \cup \{t^v_e, t^w_e\}, (E - \{e\}) \cup \{(v, t^v_e), (t^w_e, w)\})$
    {*Step B:* Update the set of lines; See Figure 3}
    $L^* \leftarrow (L - L_e) \cup (L^v_e \cup L^w_e)$, where:
    – $L^v_e = \{e^1_{e,i}\ e^2_{e,i}\ \ldots\ e^{k-1}_{e,i}\ (v, t^v_e);\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}$
    – $L^w_e = \{(t^w_e, w)\ e^{k+1}_{e,i}\ \ldots\ e^{|l_{e,i}|}_{e,i};\ k = p(e, l_{e,i}),\ i = 1, 2, \ldots, |L_e|\}$
    {*Step C:* Recursive calls.}
    if (G^* is connected) **then**
      `Rec-Draw(`G^*,L^*`)`
    else
      $G^*_1,\ G^*_2 \leftarrow$ the connected components of G^*;
      $L(G^*_i) \leftarrow$ lines of L^* induced by G^*_i;
      `Rec-Draw(`G^*_1,$L(G^*_1)$`)`;
      `Rec-Draw(`G^*_2,$L(G^*_2)$`)`;
    {*Step D:* Bundle connection.}
    while ($\texttt{BND}(t^v_e) \neq \emptyset$) **do**
      `/*` $\texttt{BND}(t^v_e)$ `and` $\texttt{BND}(t^w_e)$ `obtained from step C.2 of Draw-Base */`
      1. Connect equal bundles of $\texttt{BND}(t^v_e)$ and $\texttt{BND}(t^w_e)$;
      2. Remove connected bundles from $\texttt{BND}(t^v_e)$ and $\texttt{BND}(t^w_e)$;
      3. Split the largest bundle b_{max} of $\texttt{BND}(t^v_e) \cup \texttt{BND}(t^w_e)$;
      4. Propagate the splitting of b_{max} to all stations that it traverses.;
}

Algorithm 3: 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 represantations.}
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 $(indegree(u) == 0)$ **then** `/* Case (a) */`
    Sort in ascending order the lines that originate from u lexicographically;
  else `/* Case (b) */`
    $k \leftarrow indegree(u)$;
    $e_u^1 \dots e_u^k \leftarrow$ incoming edges of u, s.t. $e_u^i = (u_i, u)$ & $\texttt{EN}(e_u^i) < \texttt{EN}(e_u^j), \forall i < j$;
    $L_u^i \leftarrow$ lines that enter u from edge e_u^i in order they exit the right side of u_i;
    {*Step C.1:* Determine the order L_{in}^u of the lines along the left side of u}
    $L_{in}^u \leftarrow \emptyset$;
    for $i = 1$ **to** k **do**
      Stably sort the lines of L_u^i based on the numbering of the edges they use when exit u and add them to the end of L_{in}^u;
    {*Step C.2:* Determine the order L_{out}^u of the lines along the right side of u}
    $L_{out}^u \leftarrow$ Stably sort the lines of L_{in}^u based on the numbering of the edges they use when exit station u.