Line Crossing Minimization on Metro Maps*

Michael A. Bekos[1], Michael Kaufmann[2], Katerina Potika[1], and Antonios Symvonis[1]

[1] National Technical University of Athens, School of Applied Mathematics & Physical Sciences, 15780 Zografou, Athens, Greece
mikebekos@math.ntua.gr, epotik@cs.ntua.gr, symvonis@math.ntua.gr
[2] University of Tübingen, Institute for Informatics, Sand 13, 72076 Tübingen, Germany
mk@informatik.uni-tuebingen.de

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 line which connects them, whereas the paths illustrate the lines connecting terminal stations. We call this the *metro-line crossing minimization problem (MLCM)*.

In contrast to the problem of drawing the underlying graph nicely, MLCM has received fewer attention. It was recently introduced by Benkert et. al in [4]. In this paper, as a first step towards solving MLCM in arbitrary graphs, we study path and tree networks. We examine several variations of the problem for which we develop algorithms for obtaining optimal solutions.

Keywords: Metro Maps, Crossing Minimization, Lines, Paths, Trees.

1 Motivation

We consider a relatively new problem that arises when drawing metro maps or public transportation networks in general. In such drawings, we are given an undirected embedded graph $G = (V, E)$, which depicts the structure of the underlying network. In the case of metro maps, the nodes of G correspond to the train stations whereas an edge connecting two nodes implies that there exists a railway line which connects them. The problem we consider is motivated by the fact that an edge within the underlying network may be used by several metro lines. Since crossings are often considered as the main source of confusion in a visualization, we want to draw the lines along the edges of G, so that they cross each other as few times as possible.

* This work has been funded by the project PENED-2003. PENED-2003 is co - funded by the European Social Fund (75%) and Greek National Resources (25%).

In the graph drawing literature, the focus has been so far exclusively on drawing the underlying graph nicely and not on how to embed the bus or the metro lines along the underlying network. The latter problem was recently introduced by Benkert et. al in [4]. Following their approach, we assume that the underlying network has already received an embedding. The problem of determining a solution of the general metro-line routing problem, in which the graph drawing and line routing are solved simultaneously would be of particular interest as a second step in the process of automated metro map drawing.

2 Problem Definition

We are given an undirected embedded graph $G = (V, E)$. We will refer to G as the *underlying network*. We are also given a set $\mathcal{L} = \{l_1, l_2, \dots, l_k\}$ of simple paths of G (in the following, referred to as *lines*). Each line l_i consists of a sequence of edges $e_1 = (v_0, v_1), \dots, e_d = (v_{d-1}, v_d)$ of G. The nodes v_0 and v_d are referred to as the *terminals* of line l_i. We also denote by $|l_i|$ the length of line l_i. The main task is to draw the lines along the edges of G, so that the number of crossings among pairs of lines is minimized. We call this the *metro-line crossing minimization problem (MLCM).* Formally, the MLCM problem is defined as a tuple $(G, \mathcal{L})$, where G is the underlying network and $\mathcal{L}$ is the set of lines.

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. In general, the underlying network is an undirected graph. In this paper, as a first step towards solving MLCM problem in arbitrary graphs, we study path and tree networks.

For aesthetic reasons, we insist that the crossings between lines that traverse a node of the underlying network should not be hidden under the area occupied by that node. This implies that the relative order of the lines should not change within the nodes and therefore, all possible crossings have to take place along the edges of the underlying network.

In our approach, we assume that the nodes are drawn as rectangles, which is a quite usual convention in metro maps. Each line that traverses a node 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*. In general, we may permit tracks to all four side of the node, i.e. a line that traverses a node may use any side of it to either "enter" or "leave". This model is referred to as 4-*side* model (see Figure 1). A more restricted model referred to as 2-*side* model is the one, where all lines that traverse a node use only its left and right sides (see Figure 2). In the latter case, we only allow tracks at the left and right sides of the node. Note that a solution for the MLCM problem should first specify the number of tracks that enter each side of each station and, for each track, the line of $\mathcal{L}$ that uses it.

A further refinement of the MLCM problem concerns the location of the terminals at the nodes. A particularly interesting case - that arises under the 2-side model - is the one where the lines that terminate at a station occupy its

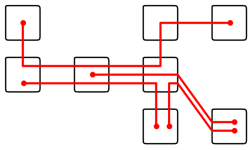
Fig. 1. 4-side model

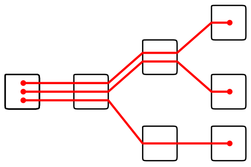
Fig. 2. 2-side model

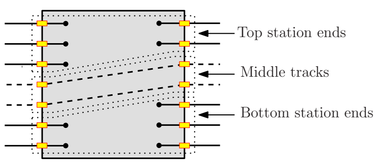

Fig. 3. Station ends, middle tracks

topmost and bottommost tracks, in the following referred to as *top* and *bottom station ends*, respectively. The remaining tracks on the left and right sides of the station are referred to as *middle tracks* and are occupied by the lines that traverse the station. Figure 3 illustrates the notions of station ends and middle tracks on the left and right sides of a station (solid lines correspond to lines that terminate, whereas the dashed lines correspond to lines that traverse the station). Based on these we introduce 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 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.

2.1 Related Literature

The problem of drawing a graph with a minimum number of crossings has been extensively studied in the graph drawing literature. For a quick survey refer to [2] and [6]. However, in the problems we study in this paper 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.

This problem was recently introduced by Benkert et. al in [4]. In their work, they proposed a dynamic-programming based algorithm that runs in $O(n^2)$ time for the *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. 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 do not terminate either at u or 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 intersecting 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 appropriate insertion orders so that the number of pairs of intersecting lines is minimized. However, Benkert et. al [4] do not address the case of larger graphs and they leave as an open problem the case where the lines that terminate at a station occupy its station ends.

For the latter problem, Asquith et al. [1] proposed an integer linear program, which always determines an optimal solution regardless the type of the underlying network. They mention that their approach can be generalized to support the case where the set of the lines consists of subgraphs of the underlying network of maximum degree 3.

A closely related problem to the one we consider is the problem of drawing a metro map nicely, widely known as *metro map layout problem*. Hong et al. [5] implemented five methods for drawing metro maps using modifications of spring-based graph drawing algorithms. Stott and Rodgers [9] have approached the problem by using a hill climbing multi-criteria optimization technique. The quality of a layout is a weighted sum over five metrics that were defined for evaluating the niceness of the resulting drawing. Nöllenburg and Wolff [8] specified the niceness of a metro map by listing a number of hard and soft constraints and they proposed a mixed-integer program which always determines a drawing that fulfills all hard constraints (if such exists) and optimizes a weighted sum of costs corresponding to the soft constraints.

In Section 3, we consider the MLCM problem on a path. We show that the MLCM-SE problem is NP-Hard and we present a polynomial time algorithm for the MLCM-FixedSE problem. In Section 4, we consider the MLCM problem on a tree and we present polynomial time algorithms for two variations of it. We conclude in Section 5 with open problems and future work. Due to lack of space, Theorem proofs are either sketched or omitted. Detailed proofs can be found in [3].

3 The Metro-line Crossing Minimization Problem on a Path

We first consider the case where the underlying network G is a path and its nodes are restricted to lie on a horizontal line. We adopt the 2-side model where each line uses the left side of a node to *"enter"* it and the right one to *"leave"* it. 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. So, 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-Hard, by reducing to it the *fixed linear crossing number problem* [7].

Definition 1. *Given a simple graph $G = (V, E)$, a* linear embedding *of G is a special type of embedding in which the nodes of V are placed on a horizontal line L and the edges are drawn as semicircles either above or bellow L.*

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

Masuda et al. [7] proved that it is NP-Hard to determine a linear embedding of a given graph with minimum number of crossings, even if the ordering of the nodes on L is fixed. The latter problem is referred to as *fixed linear crossing number problem.*

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

Proof. Let I be an instance of the fixed linear crossing number problem, consisting of a graph $G = (V, E)$ and a horizontal input line L, where $V = \{u_1, u_2, \ldots, u_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Without loss of generality, we assume that $u_1 < u_2 < \ldots < u_n$. 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 $E' = \{(u_{i-1}, u_i);\ 1 \le i \le n+1\}$. The set of lines $\mathcal{L}$ is partitioned into two sets $\mathcal{L}^A$ and $\mathcal{L}^B$:

- $\mathcal{L}^A$ consists of a sufficiently large number of lines (e.g. $2nm^2$ lines) connecting u_0 with u_{n+1}.
- $\mathcal{L}^B$ contains m lines $l_1, l_2, \ldots, l_m$ 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.

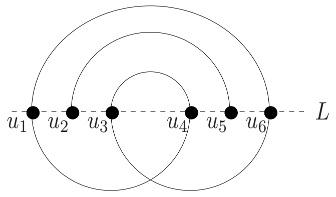

Fig. 4. A linear embedding

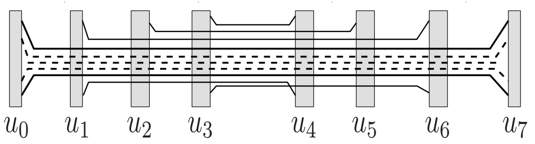

Fig. 5. An instance of MLCM-SE problem

Figures 4 and 5 illustrate the construction. First observe that all lines of $\mathcal{L}^A$ can be routed "in parallel" without any crossing among them (see Figure 5). Also observe that in an optimal solution none of the lines $l_1, l_2, \ldots, l_m$ crosses the lines of $\mathcal{L}^A$, since that would contribute a very large number of crossings. Thus, in an optimal solution each line of $\mathcal{L}^B$ has both of its terminals either at top or at bottom station ends. So, in a sense, we exclude the case where a line

$l_i \in \mathcal{L}^B$ has one of its terminals at a top station end, whereas the second one at a bottom station end. It is easy to see now that there exists an one-to-one correspondence between the crossings among the edges of I and the crossings among the lines in I', as desired. $\square$

3.1 The Metro-line Crossing Minimization Problem with Fixed Positioned Terminals

Theorem 1 implies that, unless $P = NP$, we can not efficiently determine an optimal solution of MLCM-SE problem on a path. The main reason for this is that the information whether each line terminates at the top or at the bottom station end in its terminal stations is not known in advance. In the following, we assume that this information is part of the input, which is a reasonable assumption, since terminals may represent physical locations within a station. In particular, we show that the MLCM-FixedSE problem on a path can be solved in polynomial time.

To simplify the description of our algorithm, we assume that each node u_i of the path G is adjacent to two nodes u_i^t and u_i^b, each of which will be the terminal of the lines that terminated at the top and bottom terminal tracks of node u_i, respectively[1]. In the drawing of G, u_i^t is placed directly on top of u_i (*top leg* of u_i), whereas u_i^b directly bellow it (*bottom leg* of u_i), see Figure 6a. So, instead of restricting each line to terminate at a top or at a bottom station end in its terminal stations, we will equivalently consider that it terminates to two leg nodes. We refer to this special type of graph which is implied by the addition of the leg nodes as *caterpillar with at most two legs per node.*

A caterpillar with at most two legs per node consists of two sets of nodes. The first set, denoted by V_b, contains n nodes $u_1, u_2, \ldots, u_n$ (referred to as *backbone nodes*), which form a path. In the embedding of G, these nodes are collinear and more precisely they are located on a horizontal line so that $u_1 < u_2 < \ldots < u_n$. The second set of nodes, denoted by V_l, contains n' nodes $v_1, v_2, \ldots v_{n'}$ of degree 1 (referred to as *leg nodes* or simply as *legs*) each of which is connected to one backbone node. In the embedding of G, we assume that for each backbone u one of its legs is placed directly on top of it, whereas the second one directly bellow it. Since each backbone node is adjacent to at most two legs, $n' \leq 2n$.

If v is a leg node, we will refer to its neighbor backbone node as $bn(v)$. Edges that connect backbone nodes are called *backbone edges.* Edges that connect backbone nodes with legs are called *leg edges.*

Definition 3. *Let $l \in \mathcal{L}$ be a line that connects two terminals v and v'. If v is located to the left of v' in the embedding of the underlying network, i.e. $v < v'$, then we consider v to be the* origin *of line l, whereas v' to be its* destination. *We also denote by $\mathcal{L}_i^t$ ($\mathcal{L}_i^b$) the lines that have as origin the top (bottom) leg node adjacent to backbone node u_i.*

[1] In the degenerated case, where there exists no lines terminating either at the top or bottom terminal tracks of node u_i, we assume that either u_i^t or u_i^b does not exist, respectively.

Definition 4. *Let l and l' be a pair of lines that have the same origin w and destination nodes v and v', respectively. We say that l* precedes *l', if when we start moving from w along the external face of G in counterclockwise direction we meet v before v'. The notion of precedence defines an order $\preceq$ among the lines that have the same origin, namely $l \preceq l'$, if and only if l precedes l'.*

Lemma 1. *The number of tracks in the left and right side of each backbone node that are needed in order to route all lines in $\mathcal{L}$ can be computed in $O(n+\sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

Proof. The number of tracks in the right side of the leftmost backbone node u_1 is $|\mathcal{L}_1^t| + |\mathcal{L}_1^b|$. Due to the fact that no lines have as terminal a backbone node, the same number of tracks are needed in the left side of node u_2. We index the needed tracks from top to bottom (refer to Figure 6b). We compute the number of tracks in the left side of any backbone node u_i as the number of lines originating at nodes $< u_i$ and destined for nodes $\geq u_i$. Similarly, we compute the number of tracks in the right side of any backbone node u_i as the number of lines originating at nodes $\leq u_i$ and destined for nodes $> u_i$.

Assuming that $\mathcal{L}_{u_i}$ is the set of lines that traverse a backbone node u_i, then the tracks at the left and right side of backbone node u_i can be computed in $O(|\mathcal{L}_{u_i}|)$ time, yielding to a total $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time. □

The lines of $\mathcal{L}$ are drawn incrementally by performing a left to right pass over the set of backbone nodes and by extending them from station to station with small horizontal or diagonal line segments. Therefore, each line $l \in \mathcal{L}$ is drawn as a polygonal line.

In each leg edge, that connects leg node v to $bn(v)$, we use $|\mathcal{L}_v|$ tracks indexed from right to left (refer to Figure 6b), where set $\mathcal{L}_v$ consists of the lines that either originate at or are destined for leg node v. These tracks will be used in order to route the lines that either originate at or are destined for leg node v.

In each backbone node u_i, we have to route the newly "introduced" lines, i.e. the ones that originate either at the top or at bottom leg of u_i. This procedure is illustrated in Figure 6b. We first consider the top leg node u_i^t of u_i. We sort the set $\mathcal{L}_i^t$ of the lines that originate at u_i^t in increasing order $\preceq$ of their destinations and store them in $Sort(\mathcal{L}_i^t)$. Based on this sorting we route the j-th line l in $Sort(\mathcal{L}_i^t)$ through the j-th rightmost track at the top of u_i. l is then routed to the j-th top track in the right side of u_i. We proceed by considering the bottom leg node u_i^b of u_i. Again, we sort the set $\mathcal{L}_i^b$ of the lines that originate at u_i^b in decreasing order $\preceq$ of their destinations and store them in $Sort(\mathcal{L}_i^b)$. Based on the sorting, we route the j-th line l in $Sort(\mathcal{L}_i^b)$ through the j-th rightmost track at the the bottom of u_i and then to the j-th bottom track in the right side of u_i. We then route the lines that go from the tracks of the left side to the tracks of the right side of u_i, by preserving their relative positions.

The next step is to route the lines from the right side of u_i to the left side of u_{i+1}. This is done by performing three passes over the set of tracks of the right side of u_i.

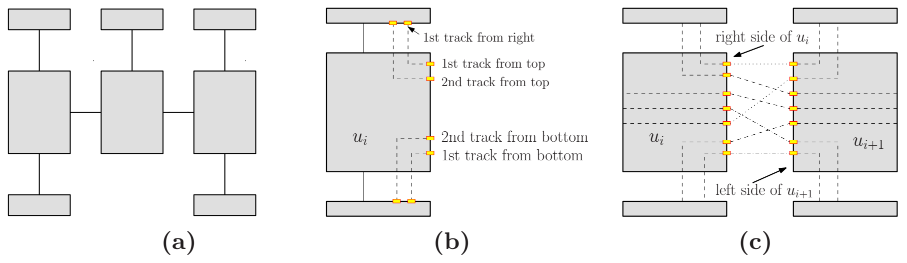

Fig. 6. (a) A caterpillar with at most 2 legs per node, (b) Introducing new lines to a station, (c) Routing lines along a backbone edge

In the first pass, we consider the tracks of the right side of u_i from top to bottom and we check whether the line l that occupies the j-th track is destined for the leg node u^t_{i+1}. In this case, we route l to the topmost available track of the right side of u_{i+1} and then to the leftmost available track in the leg edge which connects u_{i+1} with u^t_{i+1} (see the dotted lines of Figure 6c). In the second pass, we consider the remaining tracks of the right side of u_i from bottom to top and we check whether the line l that occupies the j-th track is destined for the leg node u^b_{i+1}. In this case, we route l to the bottommost available track of the right side of u_{i+1} and then to the leftmost available track in the leg edge which connects u_{i+1} with u^b_{i+1} (see the dash dotted lines of Figure 6c).

The remaining tracks of the right side of u_i are obviously occupied by the lines that are not destined for either u^t_{i+1} or for u^b_{i+1}. We consider these tracks from top to bottom and we route the line l that occupies the j-th track to the topmost available track of the right side of u_{i+1} (see the dashed lines of Figure 6c). The construction of our algorithm guarantees the following two properties:

Property of common destinations: Lines that are destined for the same top (bottom) leg node u^t_i (u^b_i) do not cross each other along the backbone edge which connects u_{i-1} with u_i.
Property of parallel routing: Two lines that both traverse a backbone node u_i (i.e. none of them are destined either for u^t_i or for u^b_i) do not cross each other along the backbone edge which connects u_{i-1} with u_i.

By combining the property of *common destinations* and the property of *parallel routing*, we easily obtain the following lemma.

Lemma 2. *In a solution produced by our algorithm the followings hold:*

(i) Two lines l and l' cross each other at most once.
(ii) Two lines l and l' with the same origin do not cross each other.
(iii) Two lines l and l' with the same destination do not cross each other.
(iv) Let l and l' be two lines that cross each other and let l (l') be destined for leg node v (v'), where v is to the left of v' in the embedding of G. Then, l and l' will cross along the backbone edge which connects u_{k-1} and u_k, where $u_k = bn(v)$.

By using Lemma 2, we can show that our algorithm produces an optimal solution, in terms of line crossings. Theorem 2 summarizes our result.

Theorem 2. *An instance $(G, \mathcal{L})$ of the MLCM-FixedSE problem on an n-node path P can be solved in $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

4 The Metro-line Crossing Minimization Problem on a Tree

In this Section, we consider the MLCM problem on a tree $T = (V, E)$, where $V = \{u_1, \ldots, u_n\}$ and $E = \{e_1, \ldots, e_{n-1}\}$. In the embedding of T, we assume that the neighbors of each node u of T are located either to the left or to the right of u. In particular, we consider a "*left-to-right tree structured network*" to represent the underlying network. In such a network, we do not allow lines which make "right-to-right" or "left-to-left" turns, which implies that all lines should be x-monotone. This assumption is motivated by the fact that a train can not make an 180° turn within a station. We seek to route all lines along the edges of T, so that the total number of crossings along the lines is minimum.

We adopt the 2-side model, where each line uses the left side of a node to *"enter"* it and the right one to *"leave"* it. We refer to the edges that are adjacent to the left (right) side of node u in the embedding of T as *incoming (outgoing) edges* of u. Since we assume that the lines are x-monotone, the notions of the origin and the destination of a line, as defined in Section 3.1, also apply in the case of line crossing minimization on "left-to-right tree structured network".

We consider the case where all terminals are located only at nodes of degree 1 and the lines can terminate at any track of their terminal stations[2].

Assuming that the edges of T are directed from left to right in the embedding of T, we first perform a topological sorting over the nodes of T. We will use this sorting later on when we route all lines along the edges of T. We proceed by numbering all nodes of T with outdegree zero[3] according to the order of appearance when moving clockwise along the external face of T starting from the first node obtained from the topological sort. Note that such a numbering is unique and we refer to it as the *Euler tour numbering* of the destination nodes.

Since the number of lines that "enter" an internal node is equal to the number of lines that "leave" it, we simply have to specify either the order of the lines that enter the node or the corresponding order when they leave it. Recall that we do not permit crossings inside the nodes. As in the preceding section, we route the lines along the edges of T incrementally. We consider the nodes of T in their topological order. This ensures that whenever we consider the next node u all of its incoming lines have already been routed up to its left neighbor nodes. We distinguish the following cases:

[2] Recall that, in the case of a path network, this problem was quite easy due to the structure of the path.

[3] Such nodes are possible line destinations.

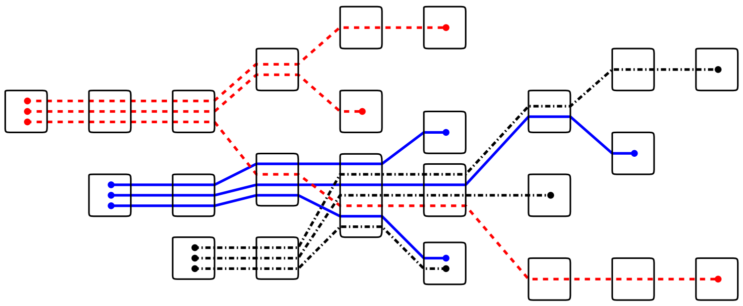

Fig. 7. A sample routing obtained from our algorithm

Case 1: $indegree(u) = 0$
If node u is of indegree zero (i.e. u is a leaf containing the origins of some lines), we simply sort the lines that originate from u based on the Euler tour numbering of their destinations in ascending order.

Case 2: $indegree(u) = 1$
We simply pass the lines from the left neighbor node of u to u without introducing any crossing (i.e. by keeping the order of the lines unchanged).

Case 3: $indegree(u) > 1$
In the case where node u is of indegree greater than one, we have to "merge" its incoming lines and thus, we may introduce crossings. We "stably merge" the incoming lines based on the Euler tour numbering of their destinations so that:

- Lines coming along the same edge do not change order.
- If two lines with the same destination come along different edges, the one coming from the topmost edge is considered to be smaller.

Figure 7 illustrates a sample routing produced by our algorithm. We use different types of lines to denote lines that originate at a common leaf node. The construction of our algorithm supports the following Lemma:

Lemma 3. *In a solution produced by our algorithm the following hold:*

(i) *Two lines l and l' cross each other at most once.*
(ii) *Two lines l and l' with the same origin do not cross each other.*
(iii) *Two lines l and l' with the same destination do not cross each other.*
(iv) *Let l and l' be two lines that cross each other. Then, l and l' will cross along their leftmost common edge.*
(v) *Let l and l' be two lines that cross each other. Then, l and l' will cross just before entering their leftmost common node.*

By using Lemma 3, we can show that our algorithm produces an optimal solution, in terms of line crossings. Theorem 3 summarizes our result.

Theorem 3. *Assuming that each line terminates at leaf nodes, an instance $(T, \mathcal{L})$ of the MLCM problem on a "left-to-right" n-node tree T can be solved in $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

4.1 The MLCM-SE and MLCM-FixedSE Problems on a Tree

Since a path can be viewed as a degenerated case of a tree, Theorem 1 implies that MLCM-SE problem on a tree is NP-Hard . However, for the MLCM-FixedSE problem we can obtain a polynomial time algorithm adopting a similar approach as the one of Section 3.1. For each node u of T we introduce at most four new nodes u_L^t, u_L^b, u_R^t and u_R^b adjacent to u. Node u_L^t (u_L^b) is placed on top (bellow) and to the left of u in the embedding of T and contains all lines that originate at u's top (bottom) station end. Similarly, node u_R^t (u_R^b) is placed on top (bellow) and to the right of u in the embedding of T and contains all lines that are destined for u's top (bottom) station end. In the case where any of the u_L^t, u_L^b, u_R^t or u_R^b does not contain any lines we ignore its existence. So, instead of restricting each line to terminate at a top or at a bottom station end in its terminal stations, we equivalently consider that it terminates to some of the newly introduced nodes. Note that the underlying network remains a tree after the introduction of the new nodes, so our algorithm can be applied in this case, too. The following Theorem summarizes our result.

Theorem 4. *An instance $(T, \mathcal{L})$ of the MLCM-FixedSE problem on a "left-to-right" n-node tree T can be solved in $O(n + \sum_{i=1}^{|\mathcal{L}|} |l_i|)$ time.*

5 Conclusions

Clearly, our work is a first step towards solving the MLCM problem and its variants in arbitrary graphs. Extending the work of Benkert et al. [4] we studied path and tree networks. However, we did not consider the case where the underlying network is an arbitrary graph. Additionally, for the case where the underlying network is a tree we only considered the case, where the terminals are located at nodes of degree 1. No results are known regarding the case where we permit terminals at internal nodes of the tree. Another line of research would be to develop approximation algorithms for the MLCM-SE problem on paths and trees. The problem of determining a solution of the general metro-line routing problem, in which the graph drawing and line routing are solved simultaneously is also of particular interest as a second step in the process of automated metro map drawing.

References

1. Asquith, M., Gudmundsson, J., Merrick, D.: An ILP for the line ordering problem. Technical Report PA006288, National ICT Australia (2007)
2. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1999)

3. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Line crossing minimization on metro maps. Technical Report WSI-2007-03, University of Tübingen (2007)
4. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transport maps. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
5. Hong, S.-H., Merrick, D., Nascimento, H.A.D.d.: The metro map layout problem. In: Churcher, N., Churcher, C. (eds.) invis.au 2004. Australasian Symposium on Information Visualisation, CRPIT, ACS, vol. 35, pp. 91–100 (2004)
6. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
7. Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. IEEE Trans. Comput. 39(1), 124–127 (1990)
8. Nöllenburg, M., Wolff, A.: A mixed-integer program for drawing high-quality metro maps. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 321–333. Springer, Heidelberg (2006)
9. Stott, J.M., Rodgers, P.: Metro map layout using multicriteria optimization. In: Proc. 8th International Conference on Information Visualisation, pp. 355–362. IEEE Computer Society, Los Alamitos (2004)