Automatic Generation of Transit Maps

Hannah Bast[1], Patrick Brosi[1], and Sabine Storandt[2]

[1] University of Freiburg (Germany)
{bast, brosi}@informatik.uni-freiburg.de
[2] JMU Würzburg (Germany)
storandt@informatik.uni-wuerzburg.de

Abstract. [TODO: abstract]

Keywords: computational geometry, graph theory, optimization

1 Introduction

– mention usefulness, mention bad google maps etc
– mention that we know of no publication adressing the MCLM or the drawing of metro maps that uses real-world data
– mention that we not only want to generate schematic maps, but maps that resemble the real-world vehicle paths as closely as possible (e.g. to serve as a map overlay)

[TODO: introduction, show some example map, either rendered by us or a manually created one to set the mood]

1.1 Related Work

In [5], the problem of minimizing intra-edge crossings in transit maps was introduced with the premises of not hiding crossings under station markers for aesthetic reasons. A polynomial time algorithm for the special case of optimizing the layout along a single edge was described. The term metro-line crossing minimization problem (MLCM) was coined in [4]. In this paper, optimal layouts for path and tree networks were investigated but arbitrary graphs were left as an open problem. In [1], [8], [2], several variants of MLCM were defined and efficient algorithms were presented for some of those, often with a restriction to planar graphs. In [3], an ILP formulation for MCLM under the periphery condition [TODO: explain here?] was introduced. The resulting ILP was shown to have a size of $\mathcal{O}(|L|^2 \cdot |E|)$ with L being the set of metro lines and E the set of edges in the derived graph. In another line of research, it was observed that many (unavoidable) crossings scattered along a single edge are also not visually pleasing, hence the idea to group crossings into block crossing was exploited [7]. The problem of minimizing the number of block crossing was proven to be NP-hard on simple graphs just like the original MCLM problem [6]. As far as we are aware there are no papers on MLCM concerned with real data or containing an experimental evaluation.

The other crucial difference of all these approaches to our work is the strong abstraction from the geometry of the transit map. In all papers listed above, the graph used for optimization is constructed upon the stations which serve as nodes and the transit lines which induce edges between those nodes. But the original embedding of the edges is not taken into account, and crossings are allowed to occur anywhere along the edges (except their endpoints). As we aim for a truthful presentation of vehicle paths in our transit map, we do not ignore the shape of the lines between stations. Instead, we introduce a new graph model for transit maps where nodes represent topology changes in the embedding and edges represent parallel line segments. This allows us to restrict crossing events to nodes (without hiding them behind station markers!). Our approach resolves some issues with the MLCM model, as e.g. the restricted applicability of some algorithms to planar graphs, and the necessity of artificial grouping of crossings (which happens naturally with our approach).

1.2 Contribution

- We introduce an approach for drawing transit maps that can be used with arbitrary input data in the GTFS format
- ... describe a simple line-sweeping approach to extract the spatial topology graph of a set of (partially overlapping) polylines
- ... describe a baseline ILP for solving the metro line crossing problem (MLCM)
- ... describe a improvied ILP for solving (MLCM) and add a simple extension that also minimizes line-separations (and subsequently, a special case of the periphery problem (MLCM-P))
- ... describe some heuristics for placing inevitable crossings in a asthecitally pleasing way
- ... describe the reduction of the minimization problem to core problem graphs which simplifies the problem

[TODO: write contributions]

1.3 Problem Definition

Let L by a set of unique transit lines and w.l.o.g. $l \in \mathbb{N}$. We call an undirected graph $T = (V, S)$ with vertices $v = (p_v, \sigma_v)$ and segments $s = (u, v, L(s), \tau_s)$ a transit graph. Each v has a position $p_v \in \mathbb{R}^2$. Furthermore we assign each segment s a polygonal chain $\tau_s = (p_1, ..., p_n), p_i \in \mathbb{R}^2$ which describes the geometrical path a segment takes through the plane. For each s, we impose $p_1 = p_u$ and $p_n = p_v$, that is the polygonal chain has to start and end at the position of the segment's start and end node. $L(s)$ is the (ordered) set of lines that traverse s. Finally, we assign each node a role $\sigma_v \in 0, 1$, where $\sigma_v = 1$ means that v is a real station node and $\sigma_v = 0$ that v is only a spatial topology node.

G can be understood as an embedding of an undirected multigraph $G = (V, E)$, where each line l traversing a segment s is transformed into a single edge. It can also be understood as an embedding of a set of $\mathbb{L}$ polygonal chains. The goal is to render T in a way that resembles a modern transit map. We chose this definition because it proved to be useful during line-ordering optimization as well as during map rendering (see Section 4). It also resembles the intuitive understanding of a transit map.

[TODO: mention in further work (or somewhere more appropriate) that it is very easy (as we have done it) to apply our approach to a directed graph.]

2 Spatial Topology Extraction

Real-world public transit data is usually given as a set of stations (with geographical positions) and a set of vehicle routes, each consisting of some meta-information (line number, color, ...) and a list of stop / time pairs representing the vehicle's schedule. In many cases, the geographical path the vehicle takes through the real world is also given. In recent years, the General Transit Feed Specification (GTFS) has become the dominant format for exchanging public transit schedules, but there are still many proprietary formats (e.g. HACON, EFA etc.) which can be converted to GTFS with moderate effort. The difference between most of the formats is mainly syntactical.

This data is essentially an embedding of a transit graph, but with two major problems:

1. there may be many (possibly thousands of) edges between two station nodes, with different paths
2. topological nodes are missing in most cases [TODO: mention possibility to store this in GTFS]

We transform this data into a well-formed transit graph G by using a simple line-sweeping algorithm to collapse (partially) similar vehicle paths into single segments. Topological nodes are inserted on-the-fly. [TODO: describe algorithm] The process can be sped up by indexing every linear segment of every vehicle path in a geometric index (for example, an R-Tree), making for a better lookup time of nearby segments.

2.1 Meta nodes

[TODO: describe meta nodes and why they can be useful]

3 Line Ordering Optimization

We want to optimize the ordering imposed on each $s \in S$ in such a way that single lines can be easily traced through the network and the rendered map looks visually pleasing. These rather subjective criteria are usually formulated as the problem if minimizing the number of line crossing in the map (see Fig. 1). This is commonly called the Metro Line Crossing Minimization Problem (MLCM). ? and ? suggest that it may improve the readability of the map if lines ending in a station only occur on the left or right side of the ordered sequence of a segment. This is called the periphery condition. MLCM under the periphery condition is called MLCM-P.

In this section, we describe a baseline integer linear program (ILP) to solve MLCM with $\mathcal{O}(|S|M^2)$ variables and $\mathcal{O}(|S|M^6)$ constraints. We introduce an improved version of this program that only needs $\mathcal{O}(|S|M^2)$ constraints and describe an extension to this ILP which solves MLCM under the line separation constraint [TODO: maybe call this MLCM-S?] MLCM-S tries to avoid the separation of lines that traverse a segment s next to each other in an adjacent segment s' (see Section 3.4). This has several benefits and also covers a special case of MLCM-P which we think is the most important.

3.1 Baseline ILP

For every segment $s \in S$, we define $|L(s)|^2$ decision variables $x_{slp} \in \{0,1\}$ where s indicates the segment, $l \in L(s)$ the line, and $p = 1, \cdots, |L(s)|$ the position/number of the edge in the segment. We want to enforce $x_{slp} = 1$ when line l is assigned to edge number p in the segment, and 0 otherwise. This can be realized as follows:

$$\forall l \in L(s) : \sum_{p=1}^{L(s)} x_{slp} = 1.$$

To assure that each edge get assigned exactly one line, we additionally need the constraints:

$$\forall p \in \{1, \cdots, |L(s)|\} : \sum_{l \in L(s)} x_{slp} = 1.$$

Avoiding crossings Let A, B be two lines belonging to a segment s over vertices v, w and both extend over w. We distinguish two cases: Either there is an adjacent segment s' with $A, B \in L(s')$ as well or A and B continue in different segments. In the first case (left), A, B induce a crossing if the position of A is smaller than the position of B in s but vice versa in s'. We introduce the decision variable $x_{ss'AB} \in \{0,1\}$, which should be 1 in case a crossing is induced and 0 otherwise. To enforce this, we create one constraint per possible crossing as follows:

$$\begin{aligned}
x_{sA1} + x_{sB2} + x_{s'A2} + x_{s'B1} - x_{ss'AB} &\leq 3 \\
x_{sA1} + x_{sB3} + x_{s'A2} + x_{s'B1} - x_{ss'AB} &\leq 3 \\
x_{sA2} + x_{sB3} + x_{s'A2} + x_{s'B1} - x_{ss'AB} &\leq 3 \\
x_{sA2} + x_{sB1} + x_{s'A1} + x_{s'B2} - x_{ss'AB} &\leq 3 \\
x_{sA3} + x_{sB2} + x_{s'A1} + x_{s'B2} - x_{ss'AB} &\leq 3 \\
x_{sA3} + x_{sB2} + x_{s'A1} + x_{s'B2} - x_{ss'AB} &\leq 3.
\end{aligned}$$

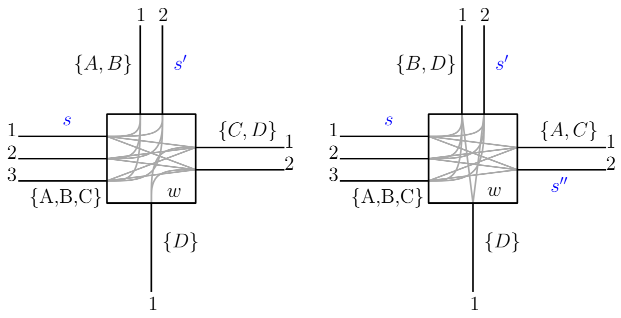

Fig. 1. Line crossings of $\{A, B\}$ in the same segments (left) and into different segments (right)

In case one of the crossings occurs, the first four variables of the corresponding constraint all have to be 1. Hence their sum equals 4 and the only way to fulfill the ≤ 3 constraint is to set $x_{ss'AB}$ to 1. The objective function then minimizes the sum over all such variables $x_{ss'AB}$.

In the second case (right), A and B split into different segments s', s''. Then the actual positions of A and B in s', s'' do not matter, but the order of s' and s'' itself. So we introduce a split crossing decision variable $x_{ss's''AB} \in \{0, 1\}$ and the following constraints:

$$\begin{aligned} x_{sA1} + x_{sB2} - x_{ss's''AB} &\leq 1 \\ x_{sA1} + x_{sB3} - x_{ss's''AB} &\leq 1 \\ x_{sA2} + x_{sB3} - x_{ss's''AB} &\leq 1. \end{aligned}$$

Likewise, $x_{ss's''AB}$ is added to the objective function.

ILP complexity Let $M = \max_{s \in S} |L(s)|$ be the maximum number of lines per segment. To get a mapping of lines to positions for each segment we need $\leq |S|M^2$ variables and $\leq 2|S|M$ constraints in total. To minimize crossings, we have to consider $\leq M^2$ pairs of lines per segment in S, and introduce a new decision variable for each such pair. That makes $\leq |S|M^2$ additional variables, which all appear in the objective function. Most constraints are introduced when both lines follow the same next segment. In that case, we create at most $\binom{M}{2}^2 < M^4$ additional constraints per line pair per segment, so $\leq |S|M^6$ in total.

In summary, there are $\mathcal{O}(|S|M^2)$ variables and $\mathcal{O}(|S|M^6)$ constraints.

3.2 Improved ILP

The $\mathcal{O}(|S|M^2)$ variables in the baseline ILP seem to be reasonable, as indeed $\Omega(|S|M^2)$ crossings could occur. But the $\mathcal{O}(|S|M^6)$ constraints are due to enumerating all possible position inversions explicitly. If on the other hand the statement *position of A on s is smaller than the position of B* would be efficiently checkable, the number of constraints could be reduced. To have such an oracle, we first modify the line-position assignment constraints. Subsequently, we use the oracle to encode inversions with fewer constraints.

Alternative line-position assignment Instead of a decision variable which encode the exact position of a line in a segment as before, we know use $x_{sl\leq p} \in \{0,1\}$ which is 1 if the position of l in s is $\leq p$ and 0 otherwise. To enforce a unique position, we use the constraints:

$$\forall l \in L(s) \forall p \in \{1, \cdots, |L(s)|-1\} : x_{sl\leq p} \leq x_{sl\leq p+1} \tag{1}$$

This ensures that the sequence can only switch from 0 to 1, and this exactly once. To make sure that at some point a 1 appears and that each position is occupied by exactly one line, we additionally introduce the following constraints:

$$\forall p \in \{1, \cdots, |L(s)|\} : \sum_{l \in L(s)} x_{sl\leq p} = p \tag{2}$$

So exactly for one line $x_{s.\leq 1} = 1$ is true, for two lines $x_{s.\leq 2} = 1$ (of which one has to fulfill $x_{s.\leq 1} = 1$) and so on.

Crossing Oracle We reconsider the example from 3.1, left. Before, we enumerated all possible positions which induce a crossing for A, B at the transition from s to s'. But it would be sufficient to have variables which tell us whether the position of A is smaller than the position of B in s, and the same for s', and then compare those variables. For a line pair (A, B) on segment s we call the respective variables $x_{sA>B}, x_{sA<b} \in \{0,1\}$. Since we introduce these variables for each line pair in s, $x_{sA<B}$ will re-appear as $x_{sB>A}$, so we only need $x_{sA>B}$. To get the desired value assignments, we add the following constraints:

$$\sum_{p=1}^{|L(s)|} x_{sA\leq p} - \sum_{p} x_{sB\leq p} + x_{sA>B}M \geq 0 \tag{3}$$

$$x_{sA>B} + x_{sB>A} = 1. \tag{4}$$

The equality constraints make sure that not both $x_{sA>B}$ and $x_{sB>A}$ can be 1. If the position of A is smaller than the position of B, then more of the variables corresponding to A are 1, hence the sum over all is higher. So if we subtract the sum for B from the sum for A and the result is ≥ 0, we know the position of A is smaller and $x_{sA>B}$ can be 0. Otherwise, the difference is negative, and we need to set $x_{sA>B}$ to 1 to fulfill the inequality. It is then indeed fulfilled for sure as the position gap can never exceed the number of lines per segment.

To finally decide if there is a crossing, we would again like to have a decision variable $x_{ss'AB} \in \{0,1\}$ which is 1 in case of a crossing and 0 otherwise – and minimize the sum of all such variables in the objective function. The constraint

$$abs(x_{sA<B} - x_{s'A<B}) - x_{ss'AB} \leq 0 \tag{5}$$

would realize this, as either $x_{sA<B} = x_{s'A<B}$ (both 0 or both 1) and then $x_{ss'AB}$ can be 0, or they are unequal and hence the absolute value of their difference is 1, enforcing $x_{ss'AB} = 1$ to fulfill the ≤ 0 condition. As absolute value computation can not be part of an ILP we use the following replacements:

$$x_{sA<B} - x_{s'A<B} - x_{ss'AB} \leq 0 \tag{6}$$
$$-x_{sA<B} + x_{s'A<B} - x_{ss'AB} \leq 0. \tag{7}$$

If the values are equal, nothing changes in the argumentation. If the values are unequal, either (6) or (7) will produce a 1 as the sum of the first two terms, enforcing $x_{ss'AB} = 1$ as desired.

Complexity of the improved ILP For the line-position assignment, we need at most $\leq |S|M^2$ variables and constraints just like in the basic formulation. For counting the crossings, we need per pair of lines per segment only a constant number of new variables and constraints as shown above, hence $\mathcal{O}(|S|M^2)$ in total.

3.3 Placement of Inevitable Crossings

Inevitable crossings are induced by the network's topology and have to occur somewhere. In the ILPs described above, the placement of these crossings largely depends on the solver's strategy. However, we found that crossings should almost never happen inside of stations, as the crossing is then hidden behind a rendered station box and the semantic aspects of it are almost entirely lost to the reader. It may also be confusing if an inevitable crossing between two lines appeared in a part of the map where only these two lines travel next to each other.

We found that favoring nodes that aren't stations ($v_\sigma = 0$) and have a higher count of incident edges as a location for crossings yields maps that appear more well-arranged. Since these criteria are invariant to the orderings imposed on S, we can control the placements by simply introducing a weighting factor $k_{ss'}$ which is added with each $x_{ss'll'}$ to the objective function. The value of $k_{ss'}$ depends on the node where s and s' meet and is set like described above.

3.4 Preventing Line Partner Separation

So far, we only optimized for the number of line crossings. This may not lead to the solution that is most visually pleasing. Consider the example given in Figure 2. The number of line crossings (1) is indeed minimized in the left example. However, the intuitive information that A and B continue together after x is lost. In the right example, this information is preserved, but the number of line crossings is now 2. A related problem can be seen in Figure [TODO: insert figure]. Both solutions have the same number of line crossings (2), but the right one clearly looks better, because the crossing is done in one pass. In both cases, we could adress the problem by punishing the separation of lines. For two adjacent

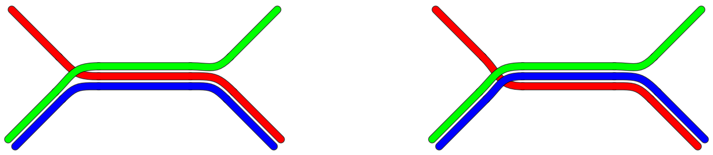

Fig. 2. Left: optimized only for line crossings, Right: optimized also for line separations, with slightly higher penalty than crossings

segments s and s' and a line pair (A, B) that continues from s to s', if A and B were placed next to each other in s (were partners in s) but not anymore in s', we want to add some penalty to the objective function. For this, we introduce a variable $x_{sA\|B} \in \{0, 1\}$. Let p_{sA} be the position of A in s and p_{sB} the position of B in s. We want $s_{sA\|B}$ to be 1 if $|p_{sA} - p_{sB}| = 0$ (if they occur next to each other) and 1 otherwise. To get the desired assignments, we add the following constraints per line pair in s:

$$\sum_{p=1}^{|L(s)|} x_{sA\leq p} - \sum_{p} x_{sB\leq p} - x_{sA\|B} M \leq 1 \tag{8}$$

$$\sum_{p=1}^{|L(s)|} x_{sB\leq p} - \sum_{p} x_{sA\leq p} - x_{sA\|B} M \leq 1. \tag{9}$$

If $p_{sA} = p_{sB}$, then the difference of the sums in both constraints is 0. If $|p_{sA} - p_{sB}| = 1$, then the sum difference in both constraints is ≤ 1. If $|p_{sA} - p_{sB}| > 1$, then either (8) or (9) enforce $x_{sA\|B} = 1$. To prevent the trivial solution where $x_{A\|B}$ is 1 for all line pairs, we introduce the following additional constraint per segment:

$$\sum_{l\in L(s)} \sum_{l'\in L(s)} x_{sl\|l'} \leq |L(s)|^2 - 3|L(s)| - 2 \tag{10}$$

as indeed the maximal number of line pairs in s (excluding lines paired with themselves) is $|L(s)|^2 - |L(s)$ and the number of $x_{sl\|l'}$ in s that are 0 is exactly $2|L(s)| - 2$ (each line is next to its two neighbors, but the first and last line only have 1 neighbor). A fixed constraint is not necessary here, however, because the lower bound is already provided by the line pair constraints (8) and (9).

Like in Section 3.2, we introduce a decision variable $x_{ss'A\|B} \in \{0, 1\}$ that should be 1 if A and B are splitted between s and s' and 0 otherwise:

$$x_{sA\|B} - x_{s'A\|B} - x_{ss'A\|B} \leq 0 \tag{11}$$

$$-x_{sA\|B} + x_{s'A\|B} - x_{ss'A\|B} \leq 0. \tag{12}$$

$x_{ss'A\|B}$ is added to the objective function. If $|L(s)| = |L(s')| = 2$, a line separation penalty is not needed for A and B because they will always be next to each other. If only $|L(s)| = 2$, we can add $x_{s'A\|B}$ to the objective function directly (and likewise if only $|L(s')| = 2$).

[TODO: Mention the special case of the MLCM-P periphery constraint the above constraints handle out-of-the-box]

Complexity We add 1 additional constraint per segment in S and a constant number of additional constraints and variables per line pair to the ILP, so the number of variables and constraints is still in $\mathcal{O}(|S|M^2)$.

3.5 Core problem graph

An inevitable crossing (or separation) between two lines j and k can only occur if there is at least one node with one of the following two properties:

1. v is adjacent to segments $s : j \notin L(s) \wedge k \in L(s')$ and $s' : j \in L(s') \wedge k \notin L(s)$.
2. $|s \in adj(v) : j \in L(s) \wedge i \in L(s)| > 2$

Proof (sketch): if no such v_jk exists, then it holds for all nodes $v \in V$ that j and k either both travel through v on the same two segments or they don't travel together through v at all (or only one of them). In the latter case, no crossing (or separation) will occur and we can thus disregard it, which leaves us only with the former case. But if j and k travel through all nodes in which a crossing could occur together on the same two segments, then the relative order of j and k does not matter anymore - we just combine j and k into a new line i and assign j and k an arbitrary (either (j, k) or (k, j) global ordering.

We call nodes that comply to (1) for a line pair $A = \{j, k\}$ a liaison node for j and k and nodes that comply to (2) a T-node for j and k. Additionally, we call a node that is a liaison and/or T-node for at least one line pair an **intersection node** and a node in which one or more lines end a **terminus node**.

Using the above, we can now greatly simplify the input graph prior to optimization with the following steps:

1. combine lines that always occur together and for which no intersection or T-node exists into a single line with a global, arbitrary relative ordering
2. delete each node with degree 2 (adjacent to segments $s = (u, v)$ and $s' = (v, w)$) and $L(s) = L(s')$ and combine s and s' into a single new segment $ss' = (u, w)$ with $L(ss') = L(s) = L(s')$. Note that such a node can never be a terminus node or an intersection node.

Since we never delete an intersection node, these operations will not affect the optimality of the found solution. In practice, they will "move" line crossings to nodes where a new line is introduced to an existing thread of lines (liaison nodes) or to nodes where topological changes are introduced to the network (T-nodes). This has a similar effect as the heuristic described in Section 3.3. For real-world networks, it can significantly reduce the number of ILP variables and constraints.

This core problem graph can be further prepared for optimiziation by splitting it into ordering-relevant connected components using the following rules:

1. cut each segment $s = (u, v)$ with $|L(s)| = 1$ into two segments $s' = (u, v')$ and $s'' = (v'', v)$. Note that v' and v'' are not connected.
2. replace each segment $s = (u, v)$ where v is a terminus node for each $l \in L(s)$ with a segment $s' = (u, v'$ where v' is only connected to s'. Delete s from the adjaceny list of v.
3. remove each segment $s = (u, v)$ where u and v are termini for all $l \in L(s)$

The line-ordering optimization can now be run independently on each connected component of the core problem graph, reducing the size of the ILPs that have to be solved and opening the way for parallelization.

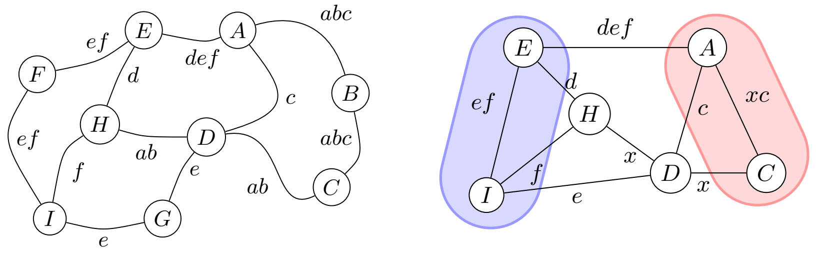

Fig. 3. Left: transit graph G with lines a, b, c, d, e, f, Right: core optimization graph with highlighted ordering-relevant connected components with more than 1 node; ab was collapsed into x.

4 Rendering

[TODO: short explanation of parallel line rendering, node front expansion for non-station nodes, explain bezier curve and explain degeneration to circle arc]

5 Evaluation

5.1 Optimization Results

5.2 Aesthetical Evaluation

6 Further work

References

1. Evmorfia Argyriou, Michael A Bekos, Michael Kaufmann, and Antonios Symvonis. Two polynomial time algorithms for the metro-line crossing minimization problem. In *International Symposium on Graph Drawing*, pages 336–347. Springer, 2008.

2. Evmorfia N Argyriou, Michael A Bekos, Michael Kaufmann, and Antonios Symvonis. On metro-line crossing minimization. *J. Graph Algorithms Appl.*, 14(1):75–96, 2010.
3. Matthew Asquith, Joachim Gudmundsson, and Damian Merrick. An ilp for the metro-line crossing problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 49–56. Australian Computer Society, Inc., 2008.
4. Michael A Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Line crossing minimization on metro maps. In *International Symposium on Graph Drawing*, pages 231–242. Springer, 2007.
5. Marc Benkert, Martin Nöllenburg, Takeaki Uno, and Alexander Wolff. Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In *International Symposium on Graph Drawing*, pages 270–281. Springer, 2006.
6. Martin Fink and Sergey Pupyrev. Metro-line crossing minimization: Hardness, approximations, and tractable cases. In *Graph Drawing*, volume 8242, pages 328–339, 2013.
7. Martin Fink and Sergey Pupyrev. Ordering metro lines by block crossings. In *International Symposium on Mathematical Foundations of Computer Science*, pages 397–408. Springer, 2013.
8. Martin Nöllenburg. An improved algorithm for the metro-line crossing minimization problem. In *International Symposium on Graph Drawing*, pages 381–392. Springer, 2009.