Efficient Generation of Geographically Accurate Transit Maps

Hannah Bast
University of Freiburg
Freiburg, Germany
bast@cs.uni-freiburg.de

Patrick Brosi
University of Freiburg
Freiburg, Germany
brosi@cs.uni-freiburg.de

Sabine Storandt
JMU Würzburg
Würzburg, Germany
storandt@informatik.uni-wuerzburg.de

ABSTRACT

We present LOOM (Line-Ordering Optimized Maps), a fully automatic generator of geographically accurate transit maps. The input to LOOM is data about the lines of a given transit network, namely for each line, the sequence of stations it serves and the geographical course the vehicles of this line take. We parse this data from GTFS, the prevailing standard for public transit data. LOOM proceeds in three stages: (1) construct a so-called line graph, where edges correspond to segments of the network with the same set of lines following the same course; (2) construct an ILP that yields a line ordering for each edge which minimizes the total number of line crossings and line separations; (3) based on the line graph and the ILP solution, draw the map. As a naive ILP formulation is too demanding, we derive a new custom-tailored formulation which requires significantly fewer constraints. Furthermore, we present engineering techniques which use structural properties of the line graph to further reduce the ILP size. For the subway network of New York, we can reduce the number of constraints from 229,000 in the naive ILP formulation to about 4,500 with our techniques, enabling solution times of less than a second. Since our maps respect the geography of the transit network, they can be used for tiles and overlays in typical map services. Previous research work either did not take the geographical course of the lines into account, or was concerned with schematic maps without optimizing line crossings or line separations.

1 INTRODUCTION

Cities with a public transit network usually have a map which illustrates the network and which is posted at all stations. Many map services also feature a transit layer where all lines and stations in an area are displayed. Such a map should satisfy the following main criteria:

(1) It should depict the topology of the network: which transit lines are offered, which stations do they serve in which order, and which transfers are possible.

(2) It should be neatly arranged and esthetically pleasing.

(3) It should reflect the geographical course of the lines, at least to some extent.

So far, such maps have been designed and drawn by hand. Concerning (3), the designers usually take some liberty, either to make the map fit into a certain format or to simplify the layout, or both.

The goal of this paper is to produce transit maps fully automatically, adhering to (3) rather strictly: within a given tolerance, the lines on the map should be drawn according to their geographical course. This rises several algorithmic challenges; in particular because the geographical course of some lines may overlap partially. These lines should then of course not be rendered on top of each other as this would obfuscate the visibility. Instead, they should be drawn next to each other. This requires to first identify overlapping parts and then to choose the line ordering in the rendered map. A bad ordering can lead to many unnecessary line crossings. Hence our goal is to find orderings that minimize these undesired crossings. As the number of possible orderings exceeds an octillion even for the transit network of medium sized cities, we need to develop efficient methods to find the best ordering in reasonable time.

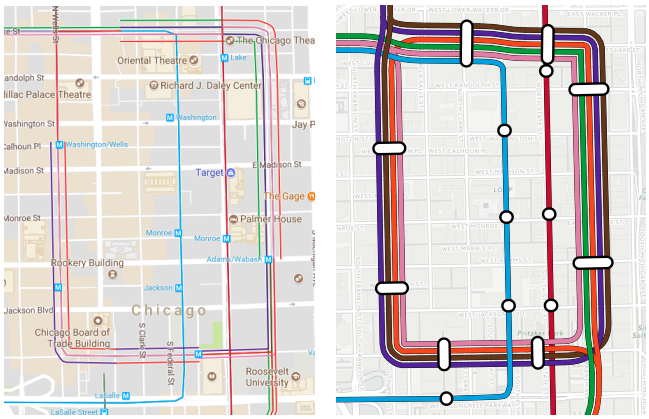

Figure 1: Left: Google transit map cutout for Chicago. Right: LOOM map for the same area.

1.1 Overview and Definitions

LOOM proceeds in three stages, which we briefly describe in the following along with some notation and terminology that will be used throughout the paper. Each stage is described in more detail in one of the following sections.

Input: The input to LOOM is a set $\mathcal{S}$ of stations and a set $\mathcal{L}$ of lines. Each station has a geographical location. Each line has a unique ID (in our examples: numbers), the sequence of stations it serves, and the geographical course between them. This data is usually provided as part of a network's GTFS feed.

Line graph construction (Sect. 2): In its first stage, LOOM computes a *line graph*. This is an undirected labeled graph $G = (V, E, L)$, where $V \supseteq \mathcal{S}$ (each station is a node, but there may be additional nodes), E is the set of edges, and each $e \in E$ is labeled with a subset $L(e) \subseteq \mathcal{L}$ of the lines. Intuitively, each edge corresponds to a segment of the network, where the same set of lines takes the same geographical course (within a certain tolerance), and there is a node wherever such a set of lines splits up in different directions. Figure 2 shows the line graph for an excerpt from the light rail network of Stuttgart. We will see that the complexity of our algorithms in Sect. 3 depends on $M = \max_{e \in E} |L(e)|$, the maximal number of lines per segment.

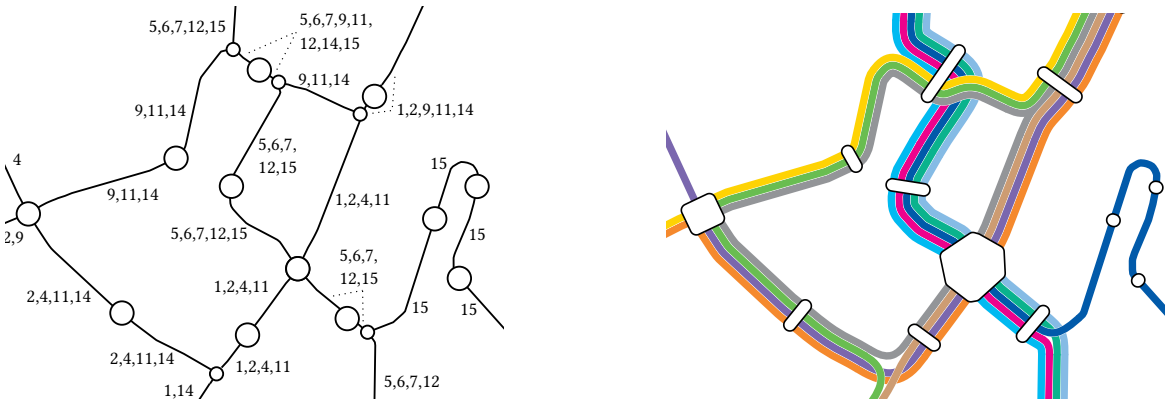

Figure 2: Left: Excerpt from a line graph which LOOM constructs for the 2015 light rail network of the city of Stuttgart from the given GTFS data. Each edge corresponds to a segment of the network where the same set of lines takes the same geographical course. Segment boundaries are often station nodes (large) but may also be intermediate nodes (small). The line ids for each segment are given in ascending order. LOOM's central optimization step computes a line ordering that determines how the lines are drawn in the map, and where line crossings and separations occur. Right: The corresponding excerpt from LOOM's transit map.

Line ordering optimization (Sect. 3): In its second stage, LOOM computes an *ordering* of $L(e)$ for each $e \in E$. This ordering determines where line crossings and separations occur, and is hence critical for the final map appearance. Previous research referred to the problem of minimizing crossings as the metro-line crossing minimization problem (MLCM), see Sect 1.3. We call a strongly related problem the metro-line node crossing minimization problem (MLNCM) with an optional line separation penalty (MLNCM-S) and formulate a concise Integer Linear Program (ILP) to solve it.

Rendering (Sect. 5): In its third stage, LOOM draws the transit map based on the line graph from stage 1 and the ordering from stage 2. Each station node v is drawn as a polygon, where each side of the polygon corresponds to exactly one incident edge of v. We call this side the *node front* of that edge at that node. The node front for an edge e has $|L(e)|$ so-called *ports* (Fig. 3). Drawing the map then amounts to connecting the ports (according to the ordering computed in stage 2) and drawing the station polygons. Figure 2, right, shows a rendered transit map after layout optimization.

1.2 Contributions

- We present a new automatic map generator, called LOOM (Line-Ordering Optimized Maps), for geographically accurate transit maps. The input is basic schedule data as provided in a GFTS feed. This is, as far as we know, the first research paper on this problem in its entirety. Previous research work considers only parts of this problem (oblivious either to the geographical course or to the order of the lines) and does not yield maps that can be used for tiles and overlays in typical map services.
- We describe a line-sweeping approach to extract the line graph from a set of (partially overlapping) vehicle trips as they occur in real-world schedule data.
- We phrase the crossing minimization problem in a novel way and provide an ILP formulation to solve it. Our new model resolves some issues with previous models, in particular, the restricted applicability of some algorithms to planar graphs, and the necessity of artificial grouping of crossings (which happens naturally with our approach).
- As a naive ILP formulation turns out to lead to impractically many constraints, we derive an alternative formulation yielding significantly smaller ILPs in theory and practice.
- We describe engineering techniques which allow to further simplify the line graph and hence lead to even smaller ILPs without compromising optimality of the final result.
- We evaluate LOOM on the transit network of six cities around the world. For each city, line graph construction, ILP solution and rendering together take less than 1 minute.
- Our maps are publicly available online[1].

1.3 Related Work

Previous research on the metro-line crossing minimization problem (MLCM), as briefly summarized in the following, typically comes without experimental evaluations and without the production of actual maps. The problem of minimizing intra-edge crossings in transit maps was introduced in [5], with the premises of not hiding crossings under station markers for esthetic reasons. A polynomial time algorithm for the special case of optimizing the layout along a single edge was described. The term MLCM was coined in [4]. In that paper, optimal layouts for path and tree networks were investigated but arbitrary graphs were left as an open problem. In [1, 2, 13], several variants of MLCM were defined and efficient algorithms were presented for some of these variants, often with a restriction to planar graphs. In [3], an ILP formulation for MLCM under the periphery condition (see Sect. 3.3) was introduced. The resulting ILP was shown to have a size of $O(|L|^2|E|)$ with L being the set of lines and E the set of edges in the derived graph. In [9], it was observed that many (unavoidable) crossings scattered along a

[1] http://loom.informatik.uni-freiburg.de

single edge are also not visually pleasing, and hence crossings were grouped into so-called block crossings. The problem of minimizing the number of block crossings was shown to be NP-hard on simple graphs just like the original MLCM problem [8]. Our adapted MLNCM problem has the same complexity as MLCM and is hence also NP-hard.

Our line graph construction is related to edge bundling. The goal of edge bundling in general networks is to group edges in order to save ink when drawing the network. Usually, the embedding of the edges is not fixed a priori but can be chosen such that many bundles occur (possibly respecting side constraints as edges being short). For example, in [11] a force-directed heuristic was described where edges attract other edges to form bundles automatically. Opposed to this, we are not allowed to embed edges arbitrarily as we want to maintain the geographical course of the vehicle trajectories. In [15], edge bundling in the context of metro line map layout was discussed, also considering orderings within the bundles to minimize crossings. But for their approach to work, the underlying graph has to fulfill a set of restrictive properties. For example, the so called *path terminal property* demands that a node in the graph cannot be an endpoint of one line and an intermediate node of another line at the same time. But this structure regularly appears in real-world instances. For example, a local train might end at the main station of a town, while a long-distance train might have this station only as an intermediate stop. Also self-intersections are forbidden which excludes instances with cyclic subway lines. With these additional properties required in [15] the problem becomes significantly easier but is no longer applicable to most real-world instances.

Another line of research focuses on drawing *schematic* metro maps, for example, by restricting the representation of transit lines to octilinear polylines [12] or Bézier Curves [7]. See also [14] for a recent survey on automated metro map layout methods. These approaches strongly abstract from the geographical course of the lines (and often also from station positions), and the minimization of line crossings or separations is not part of the problem. In particular, the resulting maps cannot be used for tiles or overlays in typical map services.

There is also some applied work on transit maps, but without publications of the details. One approach that seems to use a model similar to ours was described by Anton Dubreau in a blog post [6] although without a detailed discussion of their method. As far as we are aware there are no papers on MLCM concerned with real public transit data.

2 LINE GRAPH CONSTRUCTION

This section describes stage 1 of LOOM: given line data, construct the line graph. We assume that the data is given in the GTFS format [10]. In GTFS, each trip (that is, a concrete tour of a vehicle of a line) is given explicitly and the graph induced by this data has many overlapping edges that may (partially) share the same path.

Let e_1, e_2 be two edges in G with their geometrical paths τ_{e_1} and τ_{e_2}. We define a parametrization $\tau(t) : [0,1] \mapsto \mathbb{R}^2$ which maps the progress t to a point $p \in \mathbb{R}^2$ on τ. To decide whether a segment of τ_{e_1} is similar to a segment of τ_{e_2}, we use a simple approximation. For some distance threshold $\hat{d}$, we say $((t_1, t_2), (t'_1, t'_2))$ is a shared segment of e_1 and e_2 if

$$\forall u \in [t_1, t'_1] : \exists u' \in [t_2, t'_2] : \left\| \tau_{e_1}(u) - \tau_{e_2}(u') \right\| \leq \hat{d}. \quad (1)$$

We transform G into a line graph G' by repeatedly combining shared segments between two edges $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ into a single new edge e_{12} until no more shared segments can be found. The new path $\tau_{e_{12}}$ is averaged from the shared segments on τ_{e_1} and τ_{e_2}. Two new non-station nodes u' and v' are introduced and split e_1 and e_2 such that $e_1 = \{u_1, u'\}$, $e_2 = \{u_2, u'\}$, $e'_1 = \{v', v_1\}$, $e'_2 = \{v', v_2\}$ and $e_{12} = \{u', v'\}$. Note that the new non-station nodes v' and u' will always have a degree of 3.

To find the shared segments between τ and τ', we sweep over τ in n steps of some Δt, measuring the distance d between $\tau(i \cdot \Delta t)$ and τ' at each $i < n$ along the way. If $d \leq \hat{d}$, we start a new segment. If $d > \hat{d}$ and a segment is open, we close it. The algorithm can be made more robust against outliers by allowing d to exceed $\hat{d}$ for a number of k steps. It can be sped up by indexing every linear segment of every path in a geometric index (for example, an R-Tree).

3 LINE ORDERING OPTIMIZATION

This section describes stage 2 of LOOM, namely how to solve MLNCM: given a line graph, compute an ordering of the lines for each edge such that the total number of crossings in the final map is minimized. Contrary to the classic MLCM problem, which imposes a right and left ordering on each $L(e)$ and allows crossings to occur anywhere on e, MLNCM only imposes a single ordering on each edge and restricts crossing events to nodes. This will proof advantageous during rendering, see Sect. 5. As the set of stations $\mathcal{S}$ is only a subset of V in our model (Sect. 1.1), we can still avoid line crossings in them.

For each edge e, there are $|L(e)|!$ many orderings, therefore the total number of combinations for the whole graph is immense. We formulate an ILP to find an optimal solution. We first define a baseline ILP which explicitly considers line crossings and has $O(|E|M^2)$ variables and $O(|E|M^6)$ constraints. We then define an improved ILP with only $O(|E|M^2)$ constraints and which also considers line separations (MLNCM-S).

3.1 Baseline ILP

For every edge $e \in E$, we define $|L(e)|^2$ decision variables $x_{elp} \in \{0,1\}$ where e indicates the edge, $l \in L(e)$ indicates the line, and $p = 1, \ldots, |L(e)|$ indicates the position of the line in the edge. We want to enforce $x_{elp} = 1$ when line l is assigned to position p, and 0 otherwise. This can be realized with the following constraints:

$$\forall l \in L(e) : \sum_{p=1}^{L(e)} x_{elp} = 1. \quad (2)$$

To ensure that exactly one line is assigned to each position, we need the following additional constraints:

$$\forall p \in \{1, \ldots, |L(e)|\} : \sum_{l \in L(e)} x_{elp} = 1. \quad (3)$$

Let A, B be two lines belonging to an edge $e = \{v, w\}$ and both extend over w. We distinguish two cases: either A and B continue along the same adjacent edge e' (Fig. 3, left), or they continue along different edges e' and e'' (Fig. 3, right).

In the first case, A and B induce a crossing if the position of A is smaller than the position of B in $L(e)$, so $p_e(A) < p_e(B)$, but vice

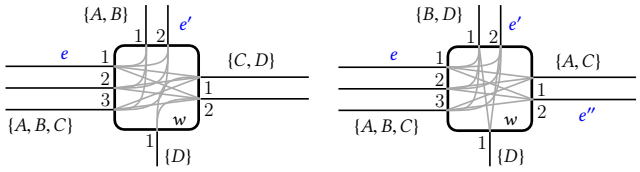

Figure 3: Example instances. Both station polygons have 4 *node fronts*, each corresponding to an incident edge. Each node front has exactly one port $(1, 2, \ldots)$ for each line traversing through its edge.

versa in $L(e')$. We introduce the decision variable $x_{ee'AB} \in \{0, 1\}$, which should be 1 in case a crossing is induced and 0 otherwise. To enforce this, we create one constraint per possible crossing. For example, a crossing would occur if we have $p_e(A) = 1$ and $p_e(B) = 2$ as well as $p_{e'}(A) = 2$ and $p_{e'}(B) = 1$. We encode this as follows:

$$x_{eA1} + x_{eB2} + x_{e'A2} + x_{e'B1} - x_{ee'AB} \leq 3. \tag{4}$$

In case the crossing occurs, the first four variables are all set to 1. Hence their sum is 4 and the only way to fulfill the ≤ 3 constraint is to set $x_{ee'AB}$ to 1. In the example given in Fig. 3, six such constraints are necessary to account for all possible crossings of the lines A and B at node w. The objective function of the ILP then minimizes the sum over all variables $x_{ee'AB}$.

In the second case, the actual positions of A and B in e' and e'' do not matter, but just the order of e' and e''. We introduce a split crossing decision variable $x_{ee'e''AB} \in \{0, 1\}$ and constraints of the form $x_{eAi} + x_{eBj} - x_{ee'e''AB} \leq 1$ for all orders of A and B at e with $i < j$ as in that case a crossing would occur. We add $x_{ee'e''AB}$ to the objective function.

For mapping lines to positions at each edge we need at most $|E|M^2$ variables and $2|E|M$ constraints. To minimize crossings, we have to consider at most M^2 pairs of lines per edge, and introduce a decision variable for each such pair. That makes at most $|E|M^2$ additional variables, which all appear in the objective function. Most constraints are introduced when two lines continue over a node in the same direction. In that case, we create no more than $\binom{M}{2}^2 < M^4$ constraints per line pair per edge, so at most $|E|M^6$ in total. In summary, we have $O(|E|M^2)$ variables and $O(|E|M^6)$ constraints.

3.2 Improved ILP Formulation

The $O(|E|M^2)$ variables in the baseline ILP seem to be reasonable, as indeed $\Omega(|E|M^2)$ crossings could occur. But the $O(|E|M^6)$ constraints are due to enumerating all possible position inversions explicitly. If we could check the statement *position of A on e is smaller than the position of B* efficiently, the number of constraints could be reduced. To have such an oracle, we first modify the line-position assignment constraints.

Instead of a decision variable encoding the exact position of a line, we now use $x_{el\leq p} \in \{0, 1\}$ which is 1 if the position of l in e is $\leq p$ and 0 otherwise. To enforce a unique position, we use the constraints:

$$\forall l \in L(e)\ \forall p \in \{1, \ldots, |L(e)| - 1\}: \quad x_{el\leq p} \leq x_{el\leq p+1}. \tag{5}$$

This ensures that the sequence can only switch from 0 to 1, 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:

$$\forall p \in \{1, \ldots, |L(e)|\}: \quad \sum\nolimits_{l \in L(e)} x_{el\leq p} = p. \tag{6}$$

So for exactly one line l, $x_{el\leq 1} = 1$, for exactly two lines l' and l'', $x_{el'\leq 2} = x_{el''\leq 2} = 1$ (where for one $l \in \{l', l''\}$, $x_{el\leq 1} = 1$) and so on.

We reconsider the example in Fig. 3, left. Before, we enumerated all possible positions which induce a crossing for A, B at the transition from e to e'. But it would be sufficient to have variables which tell us whether the position of A is smaller than the position of B in e, and the same for e', and then compare those variables. For a line pair (A, B) on edge e we call the respective variables $x_{eB<A}, x_{eA<B} \in \{0, 1\}$. To get the desired value assignments, we add the following constraints:

$$\sum\nolimits_{p=1}^{|L(e)|} x_{eA\leq p} - \sum\nolimits_{p} x_{eB\leq p} + x_{eB<A} M \geq 0 \tag{7}$$

$$x_{eB<A} + x_{eA<B} = 1. \tag{8}$$

The equality constraints make sure that not both $x_{eA<B}$ and $x_{eB<A}$ 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, and hence the sum for A 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 $x_{eB<A}$ can be 0. Otherwise, the difference is negative, and we need to set $x_{eB<A}$ 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 edge.

To decide if there is a crossing, we would again like to have a decision variable $x_{ee'AB} \in \{0, 1\}$ which is 1 in case of a crossing and 0 otherwise. The constraint

$$|x_{eA<B} - x_{e'A<B}| - x_{ee'AB} \leq 0 \tag{9}$$

realizes this, as either $x_{eA<B} = x_{e'A<B}$ (both 0 or both 1) and then $x_{ee'AB}$ can be 0, or they are not equal and hence the absolute value of their difference is 1, enforcing $x_{e'AB} = 1$. As absolute value computation cannot be part of an ILP we use the following equivalent standard replacement:

$$x_{eA<B} - x_{e'A<B} - x_{ee'AB} \leq 0 \tag{10}$$

$$-x_{eA<B} + x_{e'A<B} - x_{ee'AB} \leq 0. \tag{11}$$

For the line-position assignment, we need at most $|E|M^2$ variables and constraints just like before. For counting the crossings, we need a constant number of new variables and constraints per pair of lines per edge. Hence the total number of variables and constraints in the improved ILP is $O(|E|M^2)$.

3.3 Preventing Line Partner Separation

So far, we have only considered the number of crossings. Another relevant criterion for esthetic appeal is that "partnering" lines are drawn side by side. Fig. 4 and Fig. 5 provide two examples. We address this by punishing line separations and call this extension to our original MLNCM problem MLNCM-S. For two adjacent edges e and e' and a line pair (A, B) that continues from e to e', if A and B are placed alongside in e but not in e', we want to add a penalty

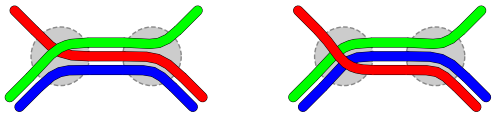

Figure 4: Minimized crossings in the left example, but the right example better indicates line pairings.

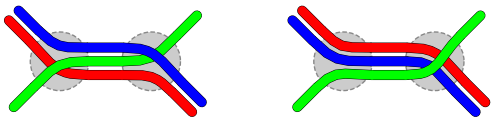

Figure 5: Both orderings have 2 crossings, but in the right example they are done in one pass.

to the objective function. For this, we add a variable $x_{eA\|B} \in \{0,1\}$ which should be 0 if $|p_e(A) - p_e(B)| = 1$ (if they are partners in e) and 1 otherwise. As $x_{eA\|B} = x_{eB\|A}$, we define a set $U(e)$ of unique line pairs such that $(l,l') \in U(e) \Rightarrow (l',l) \notin U(e)$. We add the following constraints per line pair (A,B) in $U(e)$:

$$\sum_{p=1}^{|L(e)|} x_{eA\leq p} - \sum_{p} x_{eB\leq p} - x_{eA\|B} M \leq 1 \tag{12}$$

$$\sum_{p=1}^{|L(e)|} x_{eB\leq p} - \sum_{p} x_{eA\leq p} - x_{eA\|B} M \leq 1. \tag{13}$$

If $|p_e(A) - p_e(B)| = 1$, then the sum difference is ≤ 1 and $x_{eA\|B}$ can be 0. If $|p_e(A) - p_e(B)| > 1$, then either (12) or (13) enforce $x_{eA\|B} = 1$. To prevent the trivial solution where $x_{eA\|B}$ is always 1, we add the following constraint per edge e:

$$\sum_{(l,l')\in U(e)} x_{el\|l'} \leq \binom{|L(e)|}{2} - |L(e)| - 1, \tag{14}$$

as there are $\binom{|L(e)|}{2}$ line pairs $(l,l') \in U(e)$ of which $|L(e)| - 1$ are next to each other.

Like in Sect. 3.2, we add a decision variable $x_{ee'A\|B}$ to the objective function that should be 1 if A and B are separated between e and e' and 0 otherwise:

$$x_{eA\|B} - x_{e'A\|B} - x_{ee'A\|B} \leq 0 \tag{15}$$

$$-x_{eA\|B} + x_{e'A\|B} - x_{ee'A\|B} \leq 0. \tag{16}$$

As we only add 1 constraint per edge and a constant number of constraints and variables per line pair in each edge, the total number of variables and constraints remains $O(|E|M^2)$.

Interestingly, punishing line separations also addresses a special case of the periphery condition introduced in [3]. In general, this condition holds if lines ending in a station are always drawn at the left- or rightmost position in each incident edge. For nodes with degree ≤ 2, the periphery condition is ensured in MLNCM-S (Fig. 6, left). For other nodes, however, it is not guaranteed (Fig. 6, right).

3.4 Placement of Crossings or Separations

The placement of crossings or separations may be fine-tuned by adding node-based weighting factors $w_\times(v)$ (for crossings) and $w_\|(v)$ (for separations) to the objective function to prefer nodes or to break ties. For example, $w_\times(v)$ may depend on the node degree.

As described above, we especially want to prevent crossings or separations in station nodes. This can be achieved by adding constant global weighting factors $w_{\mathcal{S}\times}$ and $w_{\mathcal{S}\|}$ to each $x_{ee'll'}$ and $x_{ee'l\|l'}$ in the objective function if l and l' continue over a node $v_s \in \mathcal{S}$. These factors have to be chosen high enough so that a crossing or separation in any other node $v \notin \mathcal{S}$ is never more expensive than in v_s. As all $w_\times(v)$ and $w_\|(v)$ appear as coefficients in the objective function, they have to be invariant to the actual line orderings. We can thus determine the maximum possible costs $\hat{w}_\times$ and $\hat{w}_\|$ prior to optimization and choose $w_{\mathcal{S}\times} = \hat{w}_\times$ and $w_{\mathcal{S}\|} = \hat{w}_\|$.

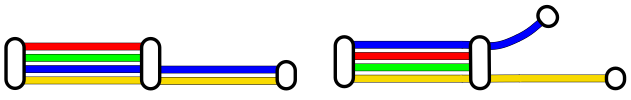

Figure 6: Left: Periphery condition guaranteed by separation penalty. Right: Periphery condition not guaranteed by separation penalty.

4 CORE GRAPH REDUCTION

It is possible to further simplify the optimization problem. We make the following observations:

Lemma 4.1. *If for some set $\mathcal{B} = \{A,B,C,\ldots\} \subseteq \mathcal{L}$ it holds for all $l \in \mathcal{B}, e \in E : l \in L(e) \Rightarrow \mathcal{B} \subseteq L(e)$, then the optimal ordering is to always bundle $A,B,C,\ldots$ next to each other with a fixed, global ordering.*

Proof. Let $L \in \mathcal{B}$ be the line that induces the minimal number of crossings and separations for some solution σ. Since all $l \in \mathcal{B}$ take the exact same path through the network, a solution can only be better than or equal to σ if it bundles all $l \neq L$ alongside L. □

Lemma 4.2. *Given an optimal ordering for each $L(e)$. We say a node v belongs to W if $\deg(v) = 2$ and for its adjacent edges e and e' the set of lines $L(e)$ is equal to $L(e')$. A crossing or a separation in some $v \in W$ can always be moved from v to a node $v' \notin W$ without negatively affecting optimality.*

Proof. We set $L^* = L(e) = L(e')$ and first consider crossings. There are two possible cases: (1) all $l \in L^*$ always occur together in each edge. Then Lemma 4.1 holds, and the ordering of $L(e)$ is the same as of $L(e')$, inhibiting any crossings in v. We can thus ignore this case. (2) Lemma 4.1 does not hold and the lines in L^* separate in some node $v' \neq v$. Then they either diverge into separate edges at v', or a subset of them ends in v'. If they diverge, the degree of v' has to be at least 3, implicating $v' \notin W$. If some (or one) of them end in v', then v' has to be adjacent to at least 2 edges e, e' with $L(e) \neq L(e')$, again implicating $v' \notin W$. Such a v' will thus indeed always exist. Under a uniform crossing penalty, we can trivially move the crossing from v to v' without affecting optimality. Under the penalty described in Sect. 3.4, optimality will also not be affected negatively, because $\deg(v)$ is always 2, implying that v is a station (Sect. 2). The same argument holds for line separations. □

Lemma 4.3. *If for some edge e all $l \in L(e)$ end in a node v or $|L(e)| = 1$, the ordering of $L(e)$ will not affect the number of orderings or separations in v.*

Proof. In the first case, no $l \in L(e)$ extends over v, so they cannot introduce any crossing or separation. In the second case, all orderings of $L(e)$ are equivalent (there is only one). □

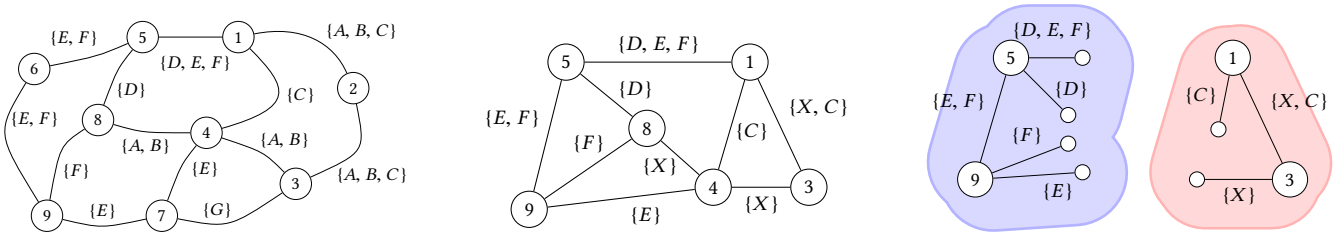

Figure 7: Left: line graph G with 7 lines. Middle: core graph of G after applying pruning rules, $\{A, B\}$ was collapsed into $\{X\}$. Right: ordering-relevant connected components of G after applying splitting rules.

Using the above, we can simplify the input graph with the following pruning rules (Fig. 7, middle):

(1) delete each node v with degree 2 and $L(e) = L(e')$, and combine the adjacent edges $e = \{u, v\}$, $e' = \{v, w\}$ into a single new edge $ee' = \{u, w\}$ with $L(ee') = L(e) = L(e')$ (Lemma 4.2).

(2) collapse lines that always occur together into a single new line k (Lemma 4.1). Weight crossings with k by the number of lines it combines to avoid distorting penalties.

(3) remove each edge $e = \{u, v\}$ where u and v are termini for all $l \in L(e)$ (Lemma 4.3).

This core graph of G may be further broken down into ordering-relevant connected components using the rules below (Fig. 7, right). The components can then be optimized separately.

(1) cut each edge $e = \{u, v\}$ with $|L(e)| = 1$ into two edges $e' = \{u, v'\}$ and $e'' = \{v'', v\}$ (Lemma 4.3).

(2) replace each edge $e = \{u, v\}$ where v has a degree > 1 and is a terminus node for each $l \in L(e)$ with an edge $e' = \{u, v'\}$ where v' is only connected to e' (Lemma 4.3).

5 RENDERING

This section describes stage 3 of LOOM: given the line graph as computed in stage 1, and a line ordering for each edge as computed in stage 2, render the actual map. We split this into four basic steps, as illustrated in Fig. 8.

In the first step (1), we make use of the fact that only a single ordering is imposed on each $L(e)$ and render each $l \in L(e)$ by perpendicular offsetting the edge's geometry τ_e by $-w\,|L(e)|/2 + w\,(p_e(l) - 1)$, where w is the desired line width. In the next step (2), we make room for the line connections between edges by expanding the node fronts (and thus the node polygon). As a stopping criteria for this expansion, we simply use a maximum distance from the node front to its original position. The line connections in the node are then rendered by connecting all port pairs (3). In our experiments, we used cubic Bézier curves, but for schematic maps a circular arc or even a straight line might be preferable.

For the station rendering (4), we found that the buffered node polygon already yields reasonable results, although with much potential for improvement. We also experimented with rotating rectangles until the total sum of the deviations between each node front orientation and the orientation of the rectangle was minimized. Both approaches can be seen in Fig. 2.

Table 1: Graph dimensions for our datasets Freiburg (FR), Dallas (DA), Chicago (CG), Stuttgart (ST), Turin (TO) and New York (NY) with extraction times from GTFS. $\mathcal{S}$ are the stations, V the graph nodes, E the graph edges and $\mathcal{L}$ the transit lines. M is the maximum number of lines per edge.

			Line graph				Core graph			
	t_{extr}	$\lvert\mathcal{S}\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{L}\rvert$	M	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{L}\rvert$	M
FR	0.7 s	74	80	81	5	4	20	21	5	4
DA	3 s	108	117	118	7	4	24	24	7	4
CG	13.5 s	143	153	154	8	6	23	24	8	6
ST	7.7 s	192	219	229	15	8	50	58	15	8
TO	4.9 s	339	398	435	14	5	91	124	14	5
NY	3.7 s	456	517	548	26	9	110	138	23	9

6 EVALUATION

We tested LOOM on the public transit schedules of six cities: Freiburg, Dallas, Chicago, Stuttgart, Turin and New York[2]. Table 1 provides the basic dimensions of each dataset and the time needed to extract the line graph.

For each dataset, we considered two versions of the line graph: the baseline graph and the core graph. For each graph, we considered three ILP variants: the baseline ILP (B), the improved ILP (I) and the improved ILP with added separation penalty (S). For each ILP, we evaluated two solvers: the GNU Linear Programming Kit (GLPK) and the COIN-OR CBC solver. As most of the datasets (except Turin) still only had one connected component after applying the splitting rules described in Sect. 4, we did not evaluate their application. Tests were run on an Intel Core i5-6300U machine with 4 cores à 2.4 GHz and 12 GB RAM. The CBC solver was compiled with multithreading support, and used with the default parameters and `threads=4`. The GLPK solver was used with the feasibility pump heuristic (`fp_heur=ON`), the proximity search heuristic (`ps_heur=ON`) and the presolver enabled (`presolve=ON`).

For each node v, the penalty for a crossing between edge pairs ($\{A, B\}$ in Figure 3, left) was $4 \cdot \deg(v)$, for other crossings ($\{A, B\}$ in Figure 3, right) it was $\deg(v)$. The line separation penalty was $3 \cdot \deg(v)$. We found that these penalties produced nicer maps than a uniform penalty. This would imply $w_{\mathcal{S}\times} = 4 \cdot \max_{v \in V} \deg(v)$ and $w_{\mathcal{S}\parallel} = 3 \cdot \max_{v \in V} \deg(v)$. However, we found that moving some crossings or separations to stations with a degree greater than 2

[2] with uncollapsed express lines

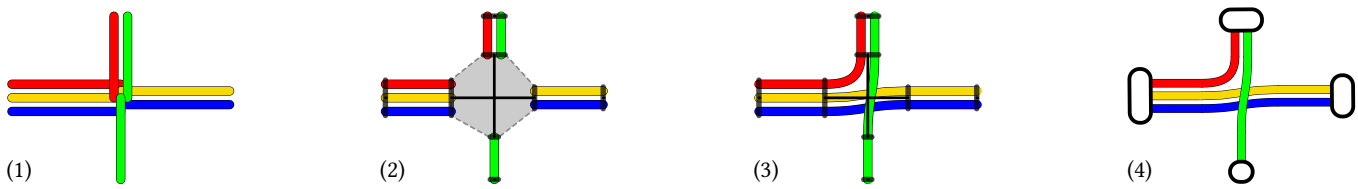

Figure 8: The four steps of rendering a given line graph: (1) render ordered lines as edges, (2) free node area, (3) render inner connections, (4) render station overlays.

Table 2: Dimensions and solution times for Chicago (CG), Stuttgart (ST), Turin (TO) and New York (NY) and our three ILPs: baseline (B), improved (I), and with line separation penalty (S), with or without reduction to the core graph. A time of — means we aborted after 12 hours. The last two columns show the number of crossings (×) and separations (||) after optimization.

		On baseline graph			On core graph				
		rows×cols	GLPK	CBC	rows×cols	GLPK	CBC	×	||
CG	B	41k×861	—	—	8.2k×266	—	47m	22	4-7
	I	1.7k×1.2k	18s	0.4s	484×352	0.4s	0.2s	22	4-7
	S	2.2k×1.4k	47m	25s	595×405	26s	3.8s	27	0
ST	B	224k×2.5k	—	—	44k×999	—	—	—	—
	I	5.1k×3.4k	—	7.1s	1.9k×1.4k	10s	1.4s	65	8-15
	S	6.6k×4.1k	—	4.5m	2.5k×1.6k	—	36s	69	3
TO	B	24k×2.1k	—	—	13k×1k	—	—	—	—
	I	4k×2.8k	32m	0.6s	1.9k×1.4k	7.1s	0.3s	78	6-10
	S	5k×3.3k	—	9.1s	2.4k×1.6k	—	4.6s	81	2
NY	B	229k×5.2k	—	—	96k×2.3k	—	—	—	—
	I	11k×7.1k	—	2.2s	4.5k×3.1k	—	0.7s	127	6-14
	S	14k×8.5k	—	2.4m	5.7k×3.7k	—	55s	132	2

yielded results that looked better. Hence, crossings in $v \in \mathcal{S}$ were punished with $w_{\mathcal{S}\times}$ if $\deg(v) = 2$ and otherwise with $3 \cdot \deg(v)$ (normal crossing) or $12 \cdot \deg(v)$ (edge-pair crossing). Similarly, in-station line separations where punished with $w_{\mathcal{S}\|}$ if $\deg(v) = 2$ and $3 \cdot \deg(v)$ otherwise. Note that Lemma 4.2 still holds because we did not change the punishment for degree 2 stations. Also note that separations were only considered in (S) and thus depended on the solver and the input order in (B) and (I).

Table 2 shows the results of the ordering optimizations for 4 of our 6 datasets. With (I), the optimal order could be found in under 1.5 seconds for each of those datasets, and in under 1 minute with (S). For medium-sized networks an optimal solution for (S) was usually found in under 5 seconds. Although the ILPs for (S) were only slightly larger than for (I), optimization on the core graph took 35 times longer on average (with CBC). (B) could not be optimized in under 12 hours for all datasets except Chicago. In general, CBC outperformed GLPK for larger datasets, sometimes dramatically. As expected, core graph reduction made the ILPs significantly smaller. On average, the number of rows decreased by 61 % and the number of columns by 59 % for (I). For (S), the decrease was 62 % and 60 %, respectively.

Table 3: Comparison of the line orderings in our maps and in manually designed schematic maps published by transportation authorities. For the schematic maps, we hand-counted the number of crossings (×) and separations (||) and calculated the score in our penalty system. T is the number of line swaps necessary to transform the line orderings in our map into those of the official map. Swaps between the same two lines on consecutive segments were only counted once.

	Official map			Our map			
	×	||	pen.	×	||	pen.	T
FR	7	1	?	7	0	96	2
DA	3	1	?	3	0	?	1
CG	27	0	?	27	0	160	1
ST	59	6	?	66	2	?	?

7 CONCLUSIONS AND FUTURE WORK

We have evaluated LOOM and shown that it produces geographically accurate transit maps. The whole pipeline took less than 1 minute for all considered inputs (including the rendering step). As the line graph construction required more time than the subsequent ILP solution for some datasets, faster algorithms for extracting the line graph would be of interest.

The ideas behind LOOM may be useful also in a non-transit scenario. For example, one closely related problem is that of wire routing in integrated-circuit design. There, stations correspond to chips and other elements (which in wire routing are indeed of polygonal form), lines correspond to wires, and the geographical course of the lines may correspond to a pre-existing wiring.

REFERENCES

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

[2] Evmorfia N Argyriou, Michael A Bekos, Michael Kaufmann, and Antonios Symvonis. 2010. On Metro-Line Crossing Minimization. *J. Graph Algorithms Appl.* 14, 1 (2010), 75–96.

[3] Matthew Asquith, Joachim Gudmundsson, and Damian Merrick. 2008. An ILP for the metro-line crossing problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*. Australian Computer Society, Inc., 49–56.

[4] Michael A Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. 2007. Line crossing minimization on metro maps. In *International Symposium on Graph Drawing*. Springer, 231–242.

[5] Marc Benkert, Martin Nöllenburg, Takeaki Uno, and Alexander Wolff. 2006. Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In *International Symposium on Graph Drawing*. Springer, 270–281.

[6] Anton Dubreau. 2016. Transit Maps: Apple vs. Google vs. Us. https://medium.com/transit-app/transit-maps-apple-vs-google-vs-us-cb3d7cd2c362. (2016).

[7] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. 2012. Drawing metro maps using Bézier curves. In *International Symposium on Graph Drawing*. Springer, 463–474.
[8] Martin Fink and Sergey Pupyrev. 2013. Metro-Line Crossing Minimization: Hardness, Approximations, and Tractable Cases.. In *Graph Drawing*, Vol. 8242. 328–339.
[9] Martin Fink and Sergey Pupyrev. 2013. Ordering metro lines by block crossings. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 397–408.
[10] GTFS. [n. d.]. General Transit Feed Specification. ([n. d.]). https://developers.google.com/transit/gtfs.
[11] Danny Holten and Jarke J Van Wijk. 2009. Force-Directed Edge Bundling for Graph Visualization. In *Computer graphics forum*, Vol. 28. Wiley Online Library, 983–990.
[12] Seok-Hee Hong, Damian Merrick, and Hugo AD do Nascimento. 2006. Automatic visualisation of metro maps. *Journal of Visual Languages & Computing* 17, 3 (2006), 203–224.
[13] Martin Nöllenburg. 2009. An improved algorithm for the metro-line crossing minimization problem. In *International Symposium on Graph Drawing*. Springer, 381–392.
[14] Martin Nöllenburg. 2014. A survey on automated metro map layout methods. *Tech. Rep.* (2014).
[15] Sergey Pupyrev, Lev Nachmanson, Sergey Bereg, and Alexander E Holroyd. 2011. Edge routing with ordered bundles.. In *Graph Drawing*, Vol. 7034. Springer, 136–147.