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 3,700 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.

CCS CONCEPTS
• Human-centered computing → Graph drawings; • Theory of computation → Integer programming;

KEYWORDS
Public Transit Network, Graph Drawing, Map generation, Graphical Optimization

ACM Reference Format:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGSPATIAL ’18, November 6–9, 2018, Seattle, WA, USA © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5889-7/18/11
https://doi.org/10.1145/3274895.3274955

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.
we need to develop efficient methods to find the best ordering in (in our examples: numbers), and information about the sequence of each station has a geographical location. Each line has a unique ID.

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. (1) given the line data, construct the line graph; (2) compute an optimal line ordering for each edge via an ILP; (3) from the line graph and the line ordering, render the map. Each stage is described in more detail in one of the following sections.

Input: The input to LOOM is a set $S$ of stations and a set $L$ of lines. Each station has a geographical location. Each line has a unique ID (in our examples: numbers), and information about 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 $V \supseteq S$ (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 $L(e) \subseteq L$ 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. Fig. 2, left 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. The line graph construction is described in Sect. 2.

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 formulate two strongly related problems: the metro-line node crossing minimization problem (MLNMC), and a variant with a line separation penalty (MLNMC-S). We formulate a concise Integer Linear Program (ILP) to solve instances of these problems.

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. 4). Drawing the map then amounts to connecting the ports (according to the ordering
computed in stage 2) and drawing the station polygons. Fig. 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 GTFS 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 15 seconds.
- Our maps are publicly available online1.

1.3 Related Work

1.3.1 Map Construction and Edge Bundling. The first step in our pipeline – the line graph construction – is closely related to map construction and edge bundling.

The goal of map construction algorithms is producing the graph of an underlying (street) network from vehicle trajectory data. There is a variety of map construction algorithms described in the literature; see [1] for an overview. For example, in [2], an incremental approach is used which starts with an empty map and incrementally updates the network graph with new trajectories. New trajectories are partially map-matched to existing graph segments with a global distance threshold and their geometries updated accordingly, while unmatched parts introduce new edges (and thus intersection nodes). The main difference between existing work (on street networks) and our approach is that our input data already represents a multigraph (with stations as intersection nodes) and is usually quite sparse.

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, like edges being short). For example, in [13] a force-directed heuristic was described where edges attract other edges to form bundles automatically. For our problem, we are not allowed to embed edges arbitrarily as we want to maintain the geographical course of the vehicle trajectories. In [17], 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 datasets. 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 [17], the problem becomes significantly easier but is no longer compatible with most real-world datasets. In contrast, our line graph construction and subsequent crossing minimization algorithms are compatible with real-world inputs of arbitrary structure.

1.3.2 Crossing Minimization. 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 [7], 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 MLCM was coined in [6]. In that paper, optimal layouts for path and tree networks were investigated but arbitrary graphs were left as an open problem. In [3, 4, 15], several variants of MLCM were defined and efficient algorithms were presented for some of these variants, often with a restriction to planar graphs. In [5], an ILP formulation for MLCM under the periphery condition (lines ending in a station must be drawn at the left- or rightmost position in incident edges, 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 [11], it was observed that crossings scattered along a 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 single graphs just like the original MLCM problem [10]. Our adapted MLNCM problem has the same complexity as MLCM and is hence also NP-hard.

1.3.3 Schematic Metro Maps. Another line of research focuses on drawing schematic metro maps, for example, by restricting the representation of transit lines to octilinear polylines [14] or Bézier Curves [9]. See [16] 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 [8] although without a detailed discussion of their method. As far as

http://loom.informatik.uni-freiburg.de
This section describes stage 1 of LOOM: given line data, construct a line graph. We assume that the data is given in the GTFS format [12]. In GTFS, each trip (that is, a concrete tour of a vehicle of a line) is given explicitly and the graph \( G^0 \) formed by all station coordinates and the trips between them has many overlapping edges that may (partially) share the same path (Fig. 3, left).

Let \( e_1, e_2 \) be two edges in \( G^0 \) with geometrical paths \( r_1 \) and \( r_2 \). For each \( r \), we define a parametrization \( p_2 (r) : [0, 1] \rightarrow \mathbb{R}^2 \) which maps the progress \( t \) to a point on \( r \) (e.g., if the length of \( r \) is 10 meters, \( p_2 (r, 0) \) returns the point we would reach after travelling on \( r \) for 5 meters). We call \((t, t'), t' \geq t\) a segment of \( e \). To decide whether a segment \((t_1, t_1')\) of \( e_1 \) is similar to a segment \((t_2, t_2')\) of \( e_2 \), we use a simple approximation. For a distance threshold \( \hat{d} = \delta (e_1, e_2) \) after collapsing an edge \( e_1 \) and \( e_2 \), we merge \( u \) and \( v \) to avoid cluttering the graph with many start and end nodes of shared segments.

As we want to avoid overlapping lines during rendering, we have to choose \( \hat{d}(e_1, e_2) \) in such a way that there will be enough space between the edges in the final line graph. Let \( w \) be the desired width of a single line in the rendered map. The definition

\[
\hat{d}(e_1, e_2) = \frac{w|L(e_1)| + w|L(e_2)|}{2}
\]

satisfies this, as we need \( w|L(e)|/2 \) map units of space on either side of \( e \) to render all \( l \in L(e) \) with width \( w \) (see Sect. 5).

We transform \( G^0 \) into a line graph \( G \) by repeatedly combining a shared segment between two edges \( e_1 = \{u_1, v_1\} \) and \( e_2 = \{u_2, v_2\} \) into a new edge \( e_{12} \) until no more shared segments can be found. The path \( r_{12} \) of \( e_{12} \) is averaged from the shared segments on \( e_1 \) and \( e_2 \), and we set \( L(e_{12}) = L(e_1) \cup L(e_2) \) (Fig. 3, right). Two new non-station nodes \( u' \) and \( v' \) which mark the beginning and end of the shared segment 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 \( u' \) and \( v' \) will always have a degree of 3. After each iteration, we obtain from \( G^0 \) a new graph \( G^{i+1} \). If the distance between a node \( u' \) added to \( G^{i+1} \) and an existing node \( v \) in \( G^i \) is smaller than \( \hat{d}(e_1, e_2) \) after collapsing an edge \( e_1 \) and \( e_2 \), we merge \( u' \) and \( v' \) to avoid cluttering the graph with many start and end nodes of shared segments.

To find the shared segments between \( e_1 \) and \( e_2 \), we sweep over \( r_1 \) in \( n \) steps of some \( \Delta t \), measuring the distance \( d \) between \( p_2 (r_1, i \cdot \Delta t) \) and \( r_2 \) at each \( i < n \) along the way. If \( d \leq \hat{d}(e_1, e_2) \), we start a new shared segment. If \( d > \hat{d}(e_1, e_2) \) and a shared segment is open, we close it. For our test datasets, we found that a \( \Delta t \) of 10 meters is usually small enough to achieve satisfying results.

The algorithm can be made more robust against outliers by allowing \( d \) to exceed \( \hat{d}(e_1, e_2) \) for a number of \( k \) steps. It can be sped up by indexing every linear segment of every path in a geometric index. Just like in previous work on incremental map construction, the results of our algorithm depend on the order in which the segments are combined. For our evaluation in Sect. 6, we used a random order.

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 edge, MLNCM only imposes a single ordering on each edge and restricts crossing events to nodes. This will prove advantageous during rendering, see Sect. 5. As the set of stations \( S \) is only a subset of \( V \) in our model (Sect. 1.1), we can still avoid line crossings in them.

### 3.1 Baseline ILP

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 \( O(|E|M^2) \) constraints. We then define an improved
ILP with only \(O(|E| M^2)\) constraints and which also considers line separations (MLNCM-S).

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:

\[
\forall l \in L (e): \sum_{p=1}^{[L(e)]} x_{elp} = 1. \quad (3)
\]

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

\[
\forall p \in \{1, \ldots, |L (e)|\}, \sum_{l \in L (e)} x_{elp} = 1. \quad (4)
\]

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. 4, left), or they continue along different edges \(e'\) and \(e''\) (Fig. 4, 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 versa in \(L (e')\). We introduce the decision variable \(x_{eAB} \in \{0, 1\}\), 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:

\[
x_{eA1} + x_{eB2} + x_{e'A2} + x_{e'B1} - x_{eAB} \leq 3. \quad (5)
\]

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 \(\leq 3\) constraint is to set \(x_{eAB}\) to 1. In the example given in Fig. 4, 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 \(x_{eAB}\).

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 \(x_{e'e''AB} \in \{0, 1\}\) and constraints of the form \(x_{eA1} + x_{eBj} - x_{e'e''AB} \leq 1\) for all orders of \(A\) and \(B\) at \(e\) with \(i < j\) as in that case a crossing would occur. We add \(x_{e'e''AB}\) to the objective function.

### 3.1.1 ILP size

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 \((M/2)^2 < M^3\) constraints per line pair per edge, so at most \(|E| M^3\) in total. In summary, we have \(O(|E| M^2)\) variables and \(O(|E| M^3)\) 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 \(O(|E| M^3)\) 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.

#### 3.2.1 Alternative line-position assignment

Instead of a decision variable encoding the exact position of a line, we now use \(x_{elp} \in \{0, 1\}\) 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:

\[
\forall l \in L (e) \forall p \in \{1, \ldots, |L (e)| - 1\}\quad x_{elp} \leq x_{elp+1}. \quad (6)
\]

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:

\[
\forall p \in \{1, \ldots, |L (e)|\}\quad \sum_{e \in L (e)} x_{elp} = p. \quad (7)
\]

So for exactly one line \(l\), \(x_{el1} = 1\), for exactly two lines \(l'\) and \(l''\), \(x_{el2} = x_{elp2} = 1\) (where for one \(l \in \{l', l''\}\), \(x_{el1} = 1\)) and so on.

#### 3.2.2 Crossing Oracle

We reconsider the example in Fig. 4, 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:

\[
\sum_{p=1}^{|L(e)|} x_{eA \leq p} - \sum_{p} x_{eB \leq p} + x_{eB < A} M \geq 0 \quad (8)
\]

\[
x_{eB < A} + x_{eA < B} = 1. \quad (9)
\]

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 \(\geq 0\), 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 \(x_{e'e''AB} \in \{0, 1\}\) which is 1 in case of a crossing and 0 otherwise. The constraint

\[
|x_{eA < B} - x_{eA < B} - x_{e'e''AB} \leq 0 \quad (10)
\]

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

\[
x_{eA < B} - x_{eA < B} - x_{e'e''AB} \leq 0 \quad (11)
\]

\[
-x_{eA < B} + x_{eA < B} - x_{e'e''AB} \leq 0. \quad (12)
\]

If the values are equal, nothing changes in the argumentation. If the values are unequal, either (11) or (12) will produce a 1 as the sum of the first two terms, enforcing \(x_{e'e''AB} = 1\) as desired.
which should be 0 if $e_1$ is not an edge. We add the following constraint per edge $e$:

$$\sum_{\rho=1}^{[L(e)]} x_{eA\rho} - \sum_{\rho=1}^{[L(e)]} x_{eB\rho} \leq x_{eA}|B| \leq 0$$

(16)

$$\sum_{\rho=1}^{[L(e)]} x_{eB\rho} - \sum_{\rho=1}^{[L(e)]} x_{eA\rho} \leq x_{eA}|B| \leq 0$$

(17)

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 $O(|E|M^2)$.

### 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. 5 and Fig. 6 provide two examples. We address this by adding the following constraints per line pair $(A, B)$ in $U(e)$.

$$\sum_{\rho=1}^{[L(e)]} x_{eA\rho} - \sum_{\rho=1}^{[L(e)]} x_{eB\rho} \leq x_{eA}|B| \leq 1$$

(13)

$$\sum_{\rho=1}^{[L(e)]} x_{eB\rho} - \sum_{\rho=1}^{[L(e)]} x_{eA\rho} \leq x_{eA}|B| \leq 1$$

(14)

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 (13) or (14) enforce $x_{eA}|B| = 1$. To prevent the trivial solution where $x_{eA}|B|$ is always 1, we add the following constraint per edge $e$:

$$\sum_{(l, l') \in U(e)} x_{eA|l|l'} \leq \frac{[L(e)]}{2} - |L(e)| - 1$$

(15)

as there are $\frac{[L(e)]}{2}$ 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 $x_{eA|l|l'}$ to the objective function that should be 1 if $A$ and $B$ are separated between $e$ and $e'$ and 0 otherwise:

$$x_{eA|B} - x_{eA|B} \leq x_{eA|B} \leq 1$$

$$-x_{eA|B} + x_{eA|B} - x_{eA|B} \leq 0$$

(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 $O(|E|M^2)$.

### 3.4 Placement of Crossings or Separations

The placement of crossings or separations may be fine-tuned by adding node-based weighting factors $w_x(v)$ (for crossings) and $w||v||$ (for separations) to the objective function to prefer nodes or to break ties. For example, $w_x(v)$ 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 $wS_X$ and $wS||$ to each $x_{eA|l|l'}$ and $x_{eA||l|l'}$ in the objective function if $l$ and $l'$ continue over a node $v_s \in S$. These factors have to be chosen high enough so that a crossing or separation in any other node $v \not\in S$ is never more expensive than in $v_s$. As all $w_x(v)$ 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 $\hat{w}_x$ and $\hat{w}||$ prior to optimization and choose $wS_X = \hat{w}_x$ and $wS|| = \hat{w}||$.

### 4 CORE GRAPH REDUCTION

It is possible to further simplify the optimization problem. In this section, we describe a set of transformations that may be applied to the line graph without affecting the global optimality of the line ordering and thus the ILP solution. In our experiments, these transformations reduced the size of the resulting ILPs by a factor between 2 and 4 and led to significantly lower solution times (see Sect. 6). We first prove Lemmata 4.1 - 4.3 and use them to derive a set of pruning and cutting rules.

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

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

Figure 7: Left: Periphery condition guaranteed by separation penalty. Right: Periphery condition not guaranteed by separation penalty.
There are two possible cases: (1) all $L$ gives an example of a core graph after applying pruning rules combines to avoid distorting penalties. (2) Lemma 4.1 does not hold and the lines in each edge. Then Lemma 4.1 holds, and the optimal ordering of $v$ separate in some node $v' \notin W$. If some (not all) 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). □

### 4.1 Pruning Rules

Using the lemmata from above, we may simplify the input line graph with the following pruning rules:

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).

We call the resulting graph the core graph of $G$. Fig. 8, middle gives an example of a core graph after applying pruning rules 1–3.

### 4.2 Cutting Rules

The core graph may then be further broken down into ordering-relevant connected components using the cutting rules below. The components can then be optimized separately and in parallel (Fig. 8, right).

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. 9.

In the first step (1), a basic skeleton of the map is rendered. We make use of the fact that only a single ordering is imposed on each $L(e)$ and draw each $l \in L(e)$ by perpendicular offsetting the segment’s geometry $\tau_e = |w|L(e)|/2 + w(p_e(l) - 1)$, where $w$ is the desired line width. As $\tau_e$ is just a piecewise linear curve, any method for offsetting (open) polygons may be used. Each drawn node $v$ now has $deg(v)$ node fronts (Fig. 9,2). The width of each

![Figure 8: 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 cutting rules.](image)

![Figure 9: 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.](image)
Table 1: Line graph dimensions for our testing datasets with extraction times from GTFS. $S$ are the stations, $V$ the graph nodes, $E$ the graph edges and $L$ the transit lines. $M$ is the maximum number of lines per edge.

|        | $t_{\text{extr}}$ | $|S|$ | $|V|$ | $|E|$ | $|L|$ | $M$  |
|--------|------------------|------|------|------|------|-----|
| Freiburg | 0.7s             | 74   | 80   | 81   | 5    | 4   |
| Dallas  | 3s               | 108  | 117  | 118  | 7    | 4   |
| Chicago | 13.5s            | 143  | 153  | 154  | 8    | 6   |
| Stuttgart | 7.7s           | 192  | 219  | 229  | 15   | 8   |
| Turin   | 4.9s             | 339  | 398  | 435  | 14   | 5   |
| New York | 3.7s            | 456  | 517  | 548  | 26   | 9   |

Table 2: Core graph dimensions for our testing datasets after applying pruning rules.

|        | $|V|$ | $|E|$ | $|L|$ | $M$  |
|--------|------|------|------|-----|
| Freiburg | 20   | 21   | 5    | 4   |
| Dallas  | 24   | 24   | 7    | 4   |
| Chicago | 23   | 24   | 8    | 6   |
| Stuttgart | 50   | 58   | 15   | 8   |
| Turin   | 91   | 124  | 14   | 5   |
| New York | 110  | 138  | 23   | 9   |

In the next step (2), we make room for the line connections between these node fronts by expanding them. As a stopping criteria for this expansion, we simply use a maximum distance from the node front to its original position.

In a third step (3), the line connections in the node are then rendered by connecting all port pairs (3). In our experiments, we used cubic Bézier curves for this, but for schematic maps a circular arc or even a straight line might be preferable.

In the last step (4), we render the stations. This is trivial for nodes of degree 1 and 2, but more complicated in large stations with multiple lines. We found that the buffered node polygon already yields reasonable results here, 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.

6 EVALUATION

We tested LOOM on the public transit schedules of six cities in Europe and the US: Freiburg, Dallas, Chicago, Stuttgart, Turin and New York. Table 1 provides the 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 three solvers: the GNU Linear Programming Kit (GLPK), the COIN-OR CBC solver and gurobi (GU). 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.

For each node $v$, the penalty for a crossing between edge pairs $(A, B)$ in Fig. 4, left) was $4 \cdot \deg(v)$, for other crossings $(A, B)$ in Fig. 4, 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 $w_{s} = 4 \cdot \max_{v \in V} \deg(v)$ and $w_{s} = 3 \cdot \max_{v \in V} \deg(v)$. However, we found that moving some crossings or separations to stations with a degree greater than 2 yielded better looking results. Hence, crossings in $v \in S$ were punished with $w_{s} = \deg(v) = 2$ and otherwise with $3 \cdot \deg(v)$ (normal crossing) or $12 \cdot \deg(v)$ (edge-pair crossing). Similarly, station line separations where punished with $w_{s} = \deg(v) = 2$ and $9 \cdot \deg(v)$ 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).

6.1 Comparison of ILP Variants

Table 4 shows the results of the ordering optimizations for 4 of our 6 datasets: Chicago, Stuttgart, Turin and New York. 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 (pre solve=ON). We used gurobi with the default parameters.

6.1.1 ILP Solution Times. With our improved ILP (I), the optimal orderings on the core graph could be found in under 50 milliseconds with gurobi, and in under 1 second with CBC, on any dataset. If line separation was also punished (S), the ILP could be solved on the core graph in under 2.5 seconds with gurobi for any dataset, and in under 1 minute with CBC. Although the ILPs for (S) were only slightly larger than for (I), optimization on the core graph took 28 times longer on average with the fastest solver.

6.1.2 Effects of Core Graph Reduction. On the baseline graph, (B) could not be optimized for all datasets except Turin with gurobi, and only after core graph reduction was a solution for Stuttgart and Chicago found in under 12 hours. As expected in Sect. 4, 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. With the fastest solver and the improved ILP (I), core graph reduction lead to speedup by a factor between 4 for Chicago, and 2 for New York. For (S), this speedup was between 22.4 for Stuttgart, and 2.5 for Turin.

6.2 Comparison to Manually Designed Maps

We also did a manual analysis to evaluate the esthetic quality of our work. For our datasets Freiburg, Dallas, Chicago and Stuttgart, we compared our automatically generated maps to the official maps.
Figure 10: Left: Google transit map cutout for Chicago. Center: Same area in our automatically generated map. Right: Official CTA map for the same area. Note the near-perfect match of the line orderings in the official map and our map.

| City     | Official Map | Our Map | $\times$ | $||$ | $\text{pen.}$ | $\times$ | $||$ | $\text{pen.}$ | $T$ |
|----------|-------------|---------|---------|-----|---------------|---------|-----|---------------|-----|
| Freiburg | 7           | 1       | 132     | 7   | 0             | 96      | 2   |               |     |
| Dallas   | 3           | 1       | 27      | 3   | 0             | 9       | 1   |               |     |
| Chicago  | 26          | 0       | 80      | 27  | 0             | 80      | 1   |               |     |
| Stuttgart| 65          | 5       | 264     | 64  | 2             | 156     | 4   |               |     |

Table 3: Comparison of the line orderings in our maps and in manually designed official maps published by transportation authorities. For the official maps, we hand-counted the number of crossings ($\times$) 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 edges were only counted once.

For each official map, we hand-counted the number of line crossings as well as the number of line separations and calculated the overall score in our penalty system. In addition, we counted the number of line swaps $T$ necessary to transform the line ordering of our map into the line ordering of the official map. Line swaps on multiple consecutive edges were only counted once. Fig. 10 gives an example of that: although we have to swap the brown and the purple line on multiple edges between stations to match the official CTA map, we only count a single, consecutive swap.

For our 4 manually evaluated datasets, we found that a surprisingly low number of line swaps was necessary to transform the line orderings found by our ILP to the line orderings of the official map. Even for the highly complex 2015 Stuttgart map, only 4 line swaps were required. This is strong evidence that our combination of penalizing line crossings and line separations closely models the aesthetics of professional, hand-drawn transit maps.

We also found that our maps always scored better or equal in our penalty system than the official maps, and that only minimal changes to the official map (missed by the designers) would be required to improve the readability. The results can be seen in Table 3. For Dallas, our ILP found a single (trivial) line swap that prevented a line separation at no cost and lowered the penalty by 66%. For Chicago, our orderings nearly match the ones in the official map, but our ILP found a solution with one additional crossing, but equivalent score. For Stuttgart, 4 line swaps could reduce both the number of crossings and the number of separations and lower the penalty by nearly 59%.

7 CONCLUSIONS AND FUTURE WORK

This paper presented a complete end-to-end method for producing geographically accurate transit maps from raw schedule data. We evaluated LOOM, a full implementation of this method, and showed that it produces geographically accurate transit maps fast. We demonstrated that our intuition of punishing both line crossings and line separations lead to results that closely resemble the aesthetics of manually designed maps.

The biggest challenge was getting the optimal line orderings in acceptable time. We have shown that with an improved formulation of our ILP and several pruning and cutting rules we could reduce
the solve time by several orders of magnitude for some datasets, compared to our initial approach. The whole pipeline (including line graph construction from GTFS schedule data, line ordering optimization and rendering) took less than 15 seconds for all considered inputs.

It may be interesting to additionally compare our ILP solution times and the final solution scores to a heuristic baseline (for example, a local search).

Since the line graph construction required more time than the subsequent ILP solution for some datasets, faster algorithms for extracting the line graph would help to further decrease the running time. It would be interesting to evaluate the adaptability of other map construction algorithms to this problem, both in terms of running time and quality.

As mentioned in Sect. 5, we see room for improvement in the rendering of station polygons. It may be necessary to enforce a local octilinearity on edges leaving stations for a cleaner look. Another open problem is that of overlapping station nodes on very small resolutions.

Both our line-ordering and rendering steps may be used with any multigraph as input and are not restricted to a geographically accurate network. It may be interesting to evaluate LOOM on schematic transit networks as well.

Lastly, 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


