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 unintuitive hardness of problem: even for a small map like the Freiburg tram network, the total number of possible edge orderings is 1.1272×10^{14}
- 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)
- mention that transit maps are not only useful for transit, but also to visualize for example flowcharts

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], [9], [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 (see Section 3.4]) 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 [8]. The problem of

minimizing the number of block crossing was proven to be NP-hard on simple graphs just like the original MCLM problem [7].

An aproach that seems to use a model similar to ours was described by Anton Dubreau in a blog post [6], although without a detailed description of their method. As far as we are aware there are no papers on MLCM concerned with real data or containing an experimental evaluation.

1.2 Contribution

We introduce an approach for automated drawing of visually pleasing transit maps that can be used with arbitrary schedule data as input. For medium sized urban rail networks, map rendering usually only takes a few seconds. A crucial difference of previous approaches to our work is the strong abstraction from the geometry of the transit map. In all of the publications 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).

We describe a simple line-sweeping approach to extract this graph from a set of (partially overlapping) vehicle trips as they occur in real-world schedule data. To solve the line-crossing problem in our model, we first describe an intuitive ILP and then develop an improved version of this program. For more aesthetic results, we introduce a line-partner separation criteria and describe an extension to our ILP that also optimizes the number of separations. To further simplify the optimization problem, we give some explicit rules to reduce our graph to an optimality-preserving core problem graph.

We also briefly discuss some heuristics for the placement of inevitable crossings or separations and describe the basic method to render a map from our optimized graph.

Our method is evaluated be running it against several real-world public transit schedules. The rendered maps have been published online[3].

1.3 Problem Definition

Let L by a set of unique transit lines. We call an undirected graph $T = (V, E)$ with vertices $v = (p_v, \sigma_v)$ and edges $e = (u, v, L(e), \tau_e)$ a transit graph. Each

[3] `http://panarea.informatik.uni-freiburg.de/gtfs-lines`

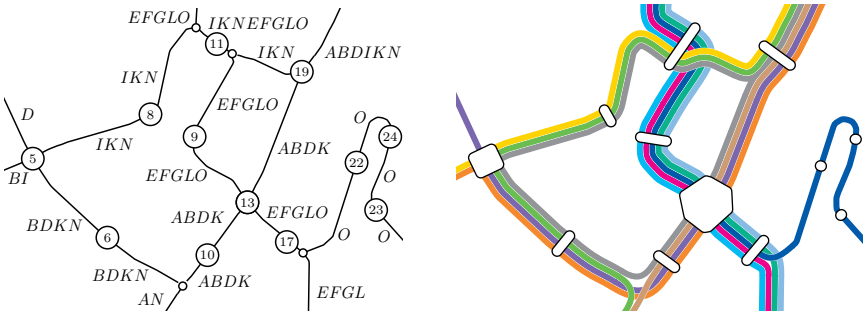

Fig. 1. Left: transit graph of the light rail network in the center of Stuttgart, with traversing lines. Topological nodes are smaller. Right: transit map rendered from graph

node has a role $\sigma_v \in 0, 1$. If $\sigma_v = 1$, then we say that v is a station node. If $\sigma_v = 0$, we call v a topology node. A topology node is a node where the topology of the underlying transit network changes. $L(e)$ is the set of lines that traverse e.

Optionally, we may assign each v a position $p_v \in \mathbb{R}^2$ and each edge e a polygonal chain $\tau_e = (p_1, ..., p_n), p_i \in \mathbb{R}^2$ which describes the geometrical path the edge takes through the plane. For each e, we then impose $p_1 = p_u$ and $p_n = p_v$.

Our goal is to render T in a way that resembles a modern transit map (Figure 1). We want to globally optimize the ordering of lines in such a way that single lines can be easily traced through the network. This is usually formulated as the problem of minimizing the number of line crossings in the map and commonly called the Metro Line Crossing Minimization Problem (MLCM).

Classic MLCM assigns each edge $e = (u, v)$ two orderings e_u and e_v for $L(e)$, one at each node [9]. Line crossings occur somewhere on e if $e_u \neq e_v$ and inevitable crossings inside of nodes are prohibited. We propose an alternative model where only a single ordering is imposed on $L(e)$, meaning the ordering remains the same throughout e. This effectively disallows crossings on edges. Instead, we explicitely allow them in nodes (and favor crossings in topological nodes, see Section 3.3). We call this the MLTCM problem, the additional T stands for topological. In our opinion, this model better resembles the way transit maps are drawn manually. In real-world maps, crossings usually occur at positions where the network topology changes and crossings inside of stations are commonly. The model also greatly simplifies the rendering process (Section 4).

MLTCM is reducible to MLCM by transforming every edge e with lines $L(e)$ into a single node n_e with $2|L(e)|$ ports. In the next step, all original nodes are replaced by edges that connect those n_e and n_f for which there existed a continued line from e to f via n in the original graph. Similarily, MLCM can be reduced to MLTCM by replacing each edge with a single node and each node with edges between all adjacent original edges with shared lines. Consequently, MLTCM is also NP-hard.

2 Network Topology Extraction

Public transit data is usually given as a set of vehicle trips. A trip is an ordered list of served stations. The vehicle's geographical path may also be given. In recent years, the General Transit Feed Specification (GTFS) has become the dominant format for exchanging public transit schedules.

Because each trip is given explicitely, the transit graph induced by this data has many overlapping edges that may (partially) share the same geographical path. The network topology is usually lost.

Let e_1, e_2 be two edges in T with their geometrical paths τ_{e_1} and τ_{e_2}. We define a parametrization $\tau(\pi) : [0, 1] \mapsto \mathbb{R}^2$ which maps the progress π to a point p_d on τ. For some distance threshold $\hat{d}$, we say $((\pi_1, \pi_2), (\pi'_1, \pi'_2))$ is a shared segment of e_1 and e_2 iff

$$\forall(\pi, \pi'), \pi \in [d_1, d'_1], \pi' \in [d_2, d'_2] : \|\tau_{e_1}(\pi) - \tau_{e_2}(\pi')\| \leq d,$$

where $\|p - p'\|$ is the euclidean distance between p and p'.

We restore the network topology and transform T into a well-formed transit graph T' by repeatedly collapsing shared segments between two edges e_1, e_2 into a single new edge e_{12} until no more shared segments can be found. If for any of the edges the start of the shared segment π lies not at the beginning of τ_e, we split $e = (u, v)$ at π into two edges e' and e'' and introduce a new topological node v' such that $e' = (u, v')$ and $e'' = (v', v)$ as well as $\tau_{e'}(1) = \tau_{e''}(0) = \tau_e(\pi)$. The same goes for the end point of a shared segment.

To find the shared segments between two paths τ and τ', we do a simple sweep over τ in n steps of some $\Delta\pi$, measuring the distance d between $\tau(i \cdot \Delta\pi)$ 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 n steps. It can be significantly sped up by indexing every linear segment of every path in a geometric index (for example, an R-Tree).

3 Line Ordering Optimization

In this section, we describe an integer linear program (ILP) to solve MLTCM with $\mathcal{O}(|E|M^2)$ variables and $\mathcal{O}(|E|M^6)$ constraints. Subsequently, we introduce an improved version with only $\mathcal{O}(|E|M^2)$ constraints and describe an extension to that ILP which solves MLTCM under the line separation constraint.

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, $l \in L(e)$ the line, and $p = 1, \cdots, |L(e)|$ the position/number

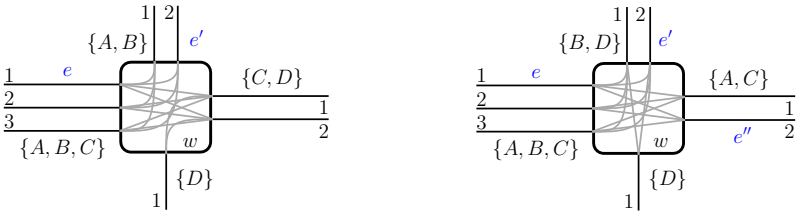

Fig. 2. Example instance.

of the line in the edge. We want to enforce $x_{elp} = 1$ when line l is assigned to edge number p in the edge, and 0 otherwise. This can be realized as follows:

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

To assure that each position gets assigned exactly one line, we additionally need the constraints:

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

Avoiding crossings Let A, B be two lines belonging to an edge $e = \{v, w\}$ and both extend over w. We distinguish two cases: Either there is an adjacent edge e' with $A, B \in L(e')$ or A and B continue in different edges, see Figure 2.

In the first case (left), A, 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 $x_{ee'AB} \in \{0, 1\}$, 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:

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

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 Figure 2, 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 (Figure 2 right), A and B split into different edges e', e''. Then the actual positions of A and B in e', e'' do not matter, but the order of e' and e'' itself. 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 $i < j$ because in that case a crossing would occur. Likewise, $x_{ee'e''AB}$ is added to the objective function.

ILP size Let $M = \max_{e \in E} |L(e)|$ be the maximum number of lines per edge. For mapping lines to positions at each edge we need $\leq |E|M^2$ variables and $\leq 2|E|M$ constraints. To minimize crossings, we have to consider $\leq M^2$ pairs of lines per edge, and introduce a decision variable for each such pair. That makes $\leq |E|M^2$ 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 $\leq \binom{M}{2}^2 < M^4$ constraints per line pair per edge, so $\leq |E|M^6$ in total. In summary, we have $\mathcal{O}(|E|M^2)$ variables and $\mathcal{O}(|E|M^6)$ constraints.

3.2 Improved ILP Formulation

The $\mathcal{O}(|E|M^2)$ variables in the baseline ILP seem to be reasonable, as indeed $\Omega(|E|M^2)$ crossings could occur. But the $\mathcal{O}(|E|M^6)$ 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.

Alternative line-position assignment Instead of a decision variable which encodes the exact position of a line in an edge as before, we know 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:

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

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:

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

So exactly for one line $x_{e.\leq 1} = 1$ is true, for two lines $x_{e.\leq 2} = 1$ (of which one has to fulfill $x_{e.\leq 1} = 1$) 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 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 $x_{eA>B}, x_{eA<B} \in \{0, 1\}$. To get the desired value assignments, we add the following constraints:

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

$$x_{eA>B} + x_{eA<B} = 1. \tag{4}$$

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 $x_{eA>B}$ can be 0. Otherwise, the difference is negative, and we need to set $x_{eA>B}$ 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 edge.

To finally 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 – and minimize the sum of all such variables in the objective function. The constraint

$$abs(x_{eA<B} - x_{e'A<B}) - x_{ee'AB} \leq 0 \tag{5}$$

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 unequal and hence the absolute value of their difference is 1, enforcing $x_{e'AB} = 1$. As absolute value computation can not be part of an ILP we use the following equivalent standard replacement:

$$x_{eA<B} - x_{e'A<B} - x_{ee'AB} \leq 0 \tag{6}$$

$$-x_{eA<B} + x_{e'A<B} - x_{ee'AB} \leq 0. \tag{7}$$

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

3.3 Placement of Inevitable Crossings

Inevitable crossings are enforced by the network's topology. In the ILPs described above, the placement of these crossings largely depends on the solver's strategy.

We want to favor crossing placements in topological nodes ($v_\sigma = 0$). We also found that favoring nodes that have a higher count of incident edges as a location for crossings yields maps that appear more well-arranged. Since all of these criteria are invariant to the orderings imposed on the $L(E)$, we can control the placements by simply introducing a weighting factor $k_{ee'}$ which is added with each $x_{ee'll'}$ to the objective function. The value of $k_{ee'}$ depends on the node where e and e' meet and is set like described above.

3.4 Preventing Line Partner Separation

So far, we only optimized the number of line crossings. However, consider the example given in Figure 3, where the ordering with more crossings looks better. A similar problem can be seen in Figure 4. In both cases, we can adress the problem by punishing the separation of lines. For two adjacent edges e and e' and a line pair (A, B) that continues from e to e', if A and B were placed

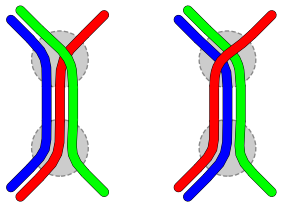

Fig. 3. Crossings are minimized in the left example (1), but the right one better indicates line pairings.

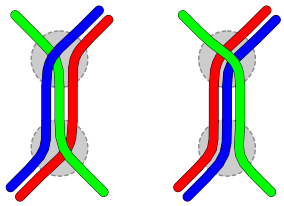

Fig. 4. Both orderings have the same number of crossings (2), but in the right one the crossing is done in one pass.

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

$$\sum_{p=1}^{|L(e)|} x_{eA\leq p} - \sum_{p} x_{eB\leq p} - x_{eA\|B}M \leq 1 \tag{8}$$

$$\sum_{p=1}^{|L(e)|} x_{eB\leq p} - \sum_{p} x_{eA\leq p} - x_{eA\|B}M \leq 1. \tag{9}$$

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

$$\sum_{l\in L(e)} \sum_{l'\in L(e)} x_{el\|l'} \leq |L(e)|^2 - 3|L(e)| - 2$$

as indeed the maximal number of line pairs in e (excluding lines paired with themselves) is $|L(e)|^2 - |L(e)|$ and the number of $x_{el\|l'}$ in e that are 0 is exactly $2|L(e)| - 2$ (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 $x_{ee'A\|B} \in \{0,1\}$ that should be 1 if A and B are splitted between e and e' and 0 otherwise:

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

$x_{ee'A\|B}$ is added to the objective function. If $|L(e)| = |L(e')| = 2$, a line separation penalty is not needed for A and B because they will always be next to each

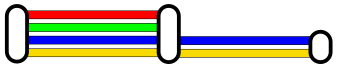

Fig. 5. Periphery condition, guaranteed by separation penalty

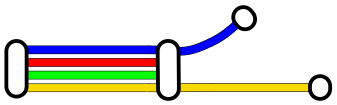

Fig. 6. Periphery condition, not guaranteed by separation penalty

other. If only $|L(e)| = 2$, we can add $x_{e'A\|B}$ to the objective function directly (and likewise if only $|L(e')| = 2$).

Interestingly, punishing line separations also adresses a special case of the periphery condition. In general, this condition holds if lines ending in some node v are always drawn at the left- or rightmost position in each edge adjacent to v [3]. For nodes with degree ≤ 2, the periphery condition is always enforced by the line separation penalty (Figure 6). For other nodes, the periphery condition may not be guaranteed anymore (Figure 6).

Complexity We add 1 additional constraint per edge in e 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 $\mathcal{O}(|E|M^2)$.

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 v with one of the following properties:

1. v is adjacent to $e : J \notin L(e) \wedge K \in L(e')$ and $e' : J \in L(e') \wedge K \notin L(e)$.
2. $|e \in adj(v) : J \in L(e) \wedge K \in L(e)| > 2$.

Proof. If no such node exists, then it holds for all $v \in V$ that J and K either both travel through v on the same two edges or they don't travel together through v at all. In the latter case, no crossing (or separation) can occur and we can thus disregard it, which leaves us only with the former case. But if J and K travel through all nodes together on the same two edges, then their relative ordering does not matter anymore - we can just combine J and K into a new line I and give them an arbitrary 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 simplify the input graph prior to optimization using the following steps:

1. collapse lines that always occur together into a single new line

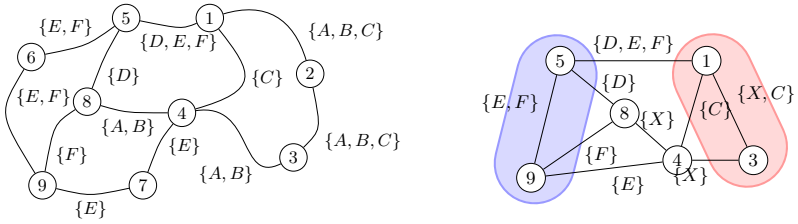

Fig. 7. Left: transit graph G with $L = \{A, B, C, D, E, F\}$, Right: core optimization graph with highlighted ordering-relevant connected components with more than 1 node; $\{A, B\}$ was collapsed into $\{X\}$.

2. delete each node with degree 2 (adjacent to edges $e = (u, v)$ and $e' = (v, w)$) and $L(e) = L(e')$ and combine e and e' into a single new edge $ee' = (u, w)$ with $L(ee') = L(e) = L(e')$. Note that such a node can never be a terminus node nor 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 liaison nodes or T-nodes. This has a similar effect as the heuristic described in Section 3.3.

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

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

The line-ordering optimization can now be run independently on each connected component of the core problem graph.

4 Rendering

We split the process of rendering a transit graph instance into four steps (Figure 8). As our graph model imposes only one line ordering per edge e, each $l \in L(e)$ can be rendered by perpendicular offsetting the edge's polyline by $-w\,|L(e)|\,/2 + w\,(p-1)$, where w is the desired width of a single line and $p \in \{1, .., |L(e)|\}$ the position of l in e (for simplicity, we disregard the spacing between lines here.) We call the perpendicular line segments of length $w\,|L(e)|$ that are formed at the beginning and end of e the node fronts of the incident node and the positions of each l on the node front the port of l. We make room for the line connections between edges by expanding these fronts.

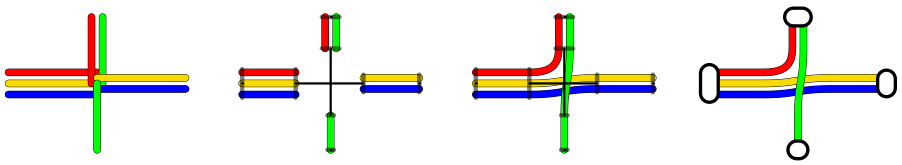

Fig. 8. From left to right: ordered lines are rendered on edges, node area is freed, inner connections are rendered, finished transit map

After the node fronts have been expanded, the line connections in the node can be rendered independently from the edges by connecting all port pairs. In our experiments, we used cubic Beziér curves, but for schematic maps a circular arc or even a straight line might be preferable.

For station rendering, we found that the (buffered) convex hull of the station's node fronts yields reasonable results, although with much potential for improvement. We also experimented with rotating rectangles until the total summed deviation between each node front orientation and the orientation of the rectangle was minimized. Both approaches can be seen in Figure 1).

5 Evaluation

To test our approach, we ran it against the public transit schedules of several cities around the world (Table 1). Each dataset was transformed into a well-formed transit graph using the method described in Section 2.

We evaluated the size of the ILP generated by the baseline approach from Section 3.1 as well as the ILP generated by the improved version from Section 3.2. We also compared the ILPs resulting from the reduced core optimization graph against those generated from the raw transit graph. Each ILP was then given to the GNU Linear Programming Kit (glpk) and the COIN-OR CBC (coin) solver. For the improved approach, we tested each dataset with or without the line separation penalty described in Section 3.4. Tests were run on an Intel Core i5-6300U machine with 4 cores à 2.4 GHz and 12 GB of RAM. The COIN solver was compiled with multithreading support. It was 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`).

Inside-station crossings added a penalty of $3n$ to the objective function, crossings between line pairs a penalty of $4n$, where $n = \deg(v)$ of the node the crossing occured in. The cost for a line separation was set to $4n$.

6 Future work

[TODO: mention thorough aesthetical evaluation via crowdsourcing?]

[†]Depends on input ordering and solving strategy

	Transit graph					Core optim. graph			
	$\lvert V_{\sigma=1}\rvert$	$\lvert V_{\sigma=0}\rvert$	$\lvert E\rvert$	$\lvert L\rvert$	M	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert L\rvert$	M
Freiburg (Stadtbahn)	76	6	83	5	4	22	23	5	4
Chicago ('L' Train)	143	10	154	8	6	23	24	8	6
Stuttgart (Stadtbahn)	192	31	235	15	8	59	71	15	8
New York (Subway)	456	60	546	26	7	124	154	23	7

Table 1. Transit graphs of testing datasets and their optimization graphs

	without graph reduction				with graph reduction					
	rows	cols	t_{glpk}	t_{coin}	rows	cols	t_{glpk}	t_{coin}	cross.	sep.†
Freiburg	4,044	354	300	**35.57**	384	126	0.9	**0.74**	5	1-2
Chicago	41,242	861	??	**>12 h**	8,272	266	??	**1,4 h**	22	4-7
Stuttgart	224,782	2,518	??	**>12 h**	44,742	1,062	??	**??**	??	??
New York	182,906	4,935	??	**>12 h**	68,452	2,206	??	**??**	??	??

Table 2. ILPs generated with baseline approach, with or without graph reduction. Solving times with glpk or COIN in "wallclock" seconds unless stated otherwise.

	without graph reduction				with graph reduction					
	rows	cols	t_{glpk}	t_{coin}	rows	cols	t_{glpk}	t_{coin}	cross.	sep.
Freiburg	654	462	0.5	**0.08**	210	162	0.3	**0.02**	5	1-2†
(sep. penalty)	906	584	4,9	**3.55**	265	188	0.5	**0.38**	7	0
Chicago	1,691	1.151	16.6	**0.51**	484	352	0.8	**0.19**	22	4-7†
	2,593	1,593	??	**23.66**	701	458	??	**6.18**	27	0
Stuttgart	5,208	3,464	??	**3.72**	2,042	1,438	0.9	**0.77**	68	12-13†
	8,280	4,970	??	**305.68**	3,185	1,996	??	**201.01**	71	6
New York	10,091	6,745	??	**2.26**	4,350	3,012	??	**1.90**	125	10-16†
	15,534	9,397	??	**326.98**	6,703	4,160	??	**58.87**	128	3

Table 3. ILPs generated with improved approach, with or without graph reduction or separation penalty. Solving times with glpk or COIN in "wallclock" seconds.

[TODO: mention station rendering, which is subpar atm]

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. Anton Dubreau. Transit Maps: Apple vs. Google vs. Us. `https://medium.com/transit-app/transit-maps-apple-vs-google-vs-us-cb3d7cd2c362`, 2016.
7. Martin Fink and Sergey Pupyrev. Metro-line crossing minimization: Hardness, approximations, and tractable cases. In *Graph Drawing*, volume 8242, pages 328–339, 2013.
8. 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.
9. Martin Nöllenburg. An improved algorithm for the metro-line crossing minimization problem. In *International Symposium on Graph Drawing*, pages 381–392. Springer, 2009.