

# Automatic Generation of Transit Maps

Hannah Bast<sup>1</sup>, Patrick Brosi<sup>1</sup>, and Sabine Storandt<sup>2</sup>

<sup>1</sup> University of Freiburg (Germany)  
`{bast, brosi}@informatik.uni-freiburg.de`  
<sup>2</sup> JMU Würzburg (Germany)  
`storandt@informatik.uni-wuerzburg.de`

**Abstract.** TODO

**Keywords:** computational geometry, graph theory, optimization

## 1 Introduction

- mention usefulness, mention bad google maps etc
- mention that we know of no publication addressing 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)

### 1.1 Problem Definition

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

$G$  can be understood as an embedding of an undirected multigraph  $G = (V, E)$ , where each line  $l$  traversing a segment  $s$  is transformed into a single edge. It can also be understood as an embedding of a set of  $\mathbb{L}$  polygonal chains.

The goal is to render  $T$  in a way that resembles a modern transit map.

We chose this definition because it proved to be useful during line-ordering optimization as well as during map rendering (see ??). It also resembles the intuitive understanding of a transit map.

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

## 1.2 Related Work

Fink and Pupyrev show that MLCM is NP-hard in [1].

## 1.3 Contribution

(TODO)

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

## 2 Spatial Topology Extraction

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

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

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

We transform this data into a well-formed transit graph  $G$  by using a simple line-sweeping algorithm to collapse (partially) similar vehicle paths into single segments. Topological nodes are inserted on-the-fly.

The process can be sped up by indexing every linear segment of every vehicle path in a geometric index (for example, an R-Tree), making for a better lookup time of nearby segments.



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

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

$$x_{sA>B} + x_{sB>A} = 1$$

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

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

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

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

$$x_{sA<B} - x_{s'A<B} - x_{ss'AB} \leq 0$$

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

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

### 3.4 Placement of Inevitable Crossings

### 3.5 Preventing Line Partner Splitting

So far, we only optimized for the number of line crossings. This may not lead to the solution that is most visually pleasing. Consider the example given in Figure 2. The number of line crossings (1) is indeed minimized in the left example. However, the intuitive information that  $A$  and  $B$  continue together after  $x$  is



**Fig. 2.** Left: optimized only for line crossings, Right: optimized also for line splittings

lost. In the right example, this information is preserved, but the number of line crossings is now 2.

A related problem can be seen in Figure TODO. Both solutions have the same number of line crossings (2), but the left one clearly looks better, because the crossing is done in one pass. In both cases, we could address the problem by punishing the separation of lines. For two adjacent segments  $s$  and  $s'$  and a line pair  $(A, B)$  that continues from  $s$  to  $s'$ , if  $A$  and  $B$  were placed next to each other in  $s$  (were partners in  $s$ ) but not anymore in  $s'$ , we want to add some penalty to the objective function. For this, we introduce a variable  $x_{sA||B} \in \{0, 1\}$ . Let  $p_{sA}$  be the position of  $A$  in  $s$  and  $p_{sB}$  the position of  $B$  in  $s$ . We want  $x_{sA||B}$  to be 1 if  $|p_{sA} - p_{sB}| = 1$  (if they occur next to each other) and 0 otherwise. To get the desired assignments, we add the following constraints per line pair in  $s$ :

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

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

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

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

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

(Mention the special case of the MLCM-P periphery constraint the above constraints handle out-of-the-box).

### 3.6 Core problem graph

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

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

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

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

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

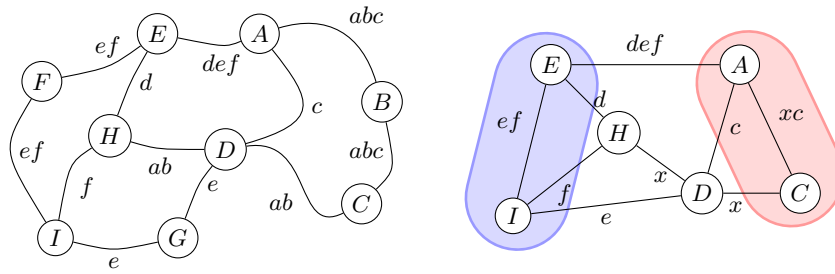
1. combine lines that always occur together in a segment and for which no intersection or T-node exists into a single line with a global, arbitrary relative ordering
2. delete each node with degree 2 (adjacent to segments  $s = (u, v)$  and  $s' = (v, w)$ ) and  $L(s) = L(s')$  and combine  $s$  and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

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

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

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



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

## 4 Rendering

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

## 5 Evaluation

### 5.1 Optimization Results

### 5.2 Aesthetical Evaluation

## References

- [1] Fink, M., Pupyrev, S.: Metro-Line Crossing Minimization: Hardness, Approximations, and Tractable Cases. *Graph Drawing*, vol. 8242, pp. 328-339. 2013.