Efficient Generation of Geographically Accurate Transit Maps

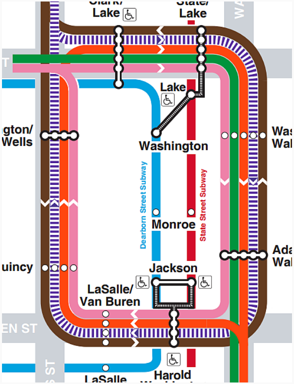
Hannah Bast[1], Patrick Brosi[1] and Sabine Storandt[2]

[1] University of Freiburg
[2] LMU Würzburg

26th ACM SIGSPATIAL - Seattle, Washington, USA

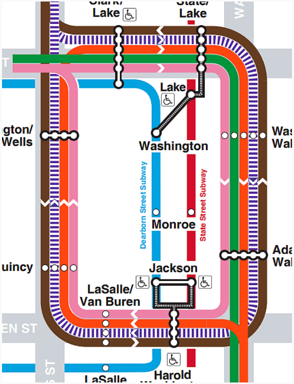
Motivation

Official CTA map

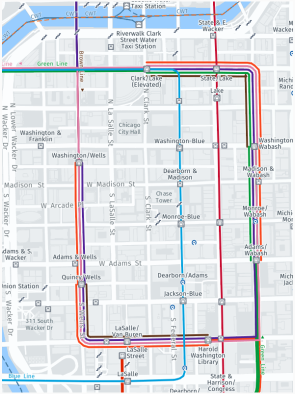


Motivation

Official CTA map

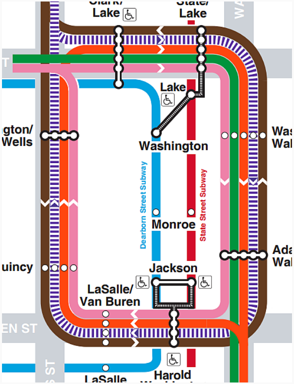


HERE

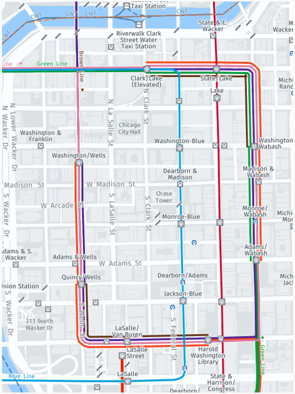


Motivation

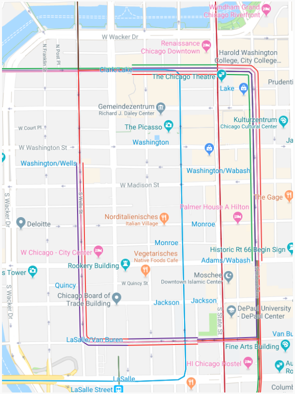
Official CTA map



HERE



Google

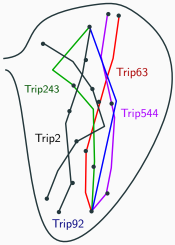


Goal

Goal: Generate these maps automatically, in high quality

Goal

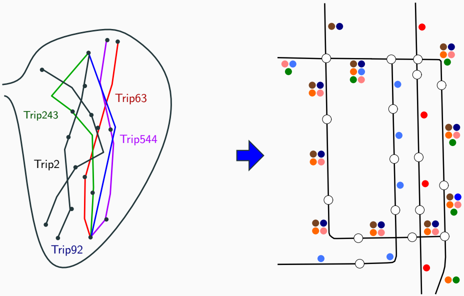
Goal: Generate these maps automatically, in high quality



”Bag of trips”
(GTFS)

Goal

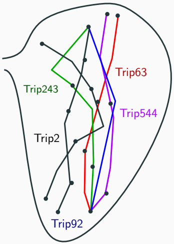
Goal: Generate these maps automatically, in high quality

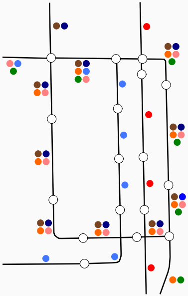


”Bag of trips”
(GTFS)

Goal

Goal: Generate these maps automatically, in high quality



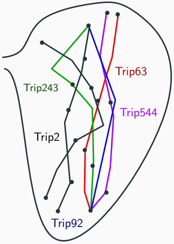


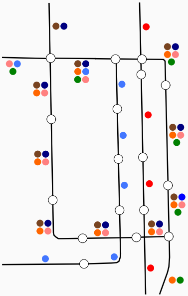
”Bag of trips”
(GTFS)

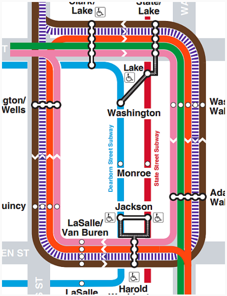
Line graph

Goal

Goal: Generate these maps automatically, in high quality





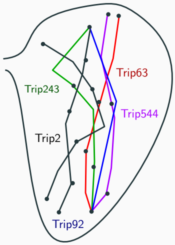


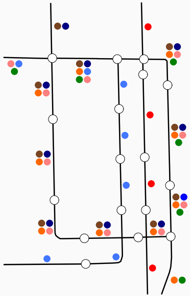
"Bag of trips"
(GTFS)

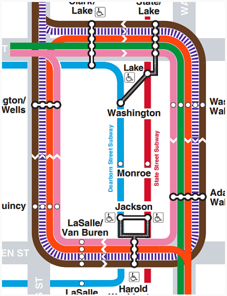
Line graph

Goal

Goal: Generate these maps automatically, in high quality







"Bag of trips" (GTFS)

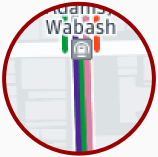
Line graph

Final map

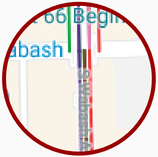
Challenges

Official

HERE

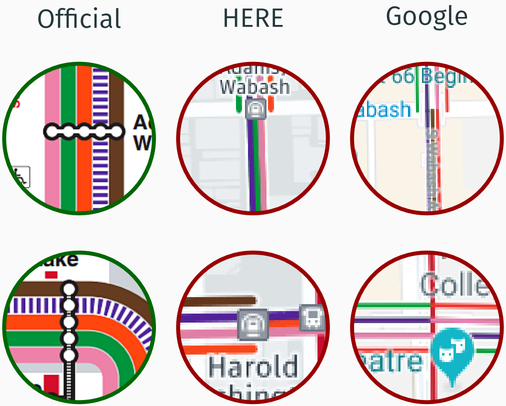


Google



I. Avoid line overlaps

Challenges

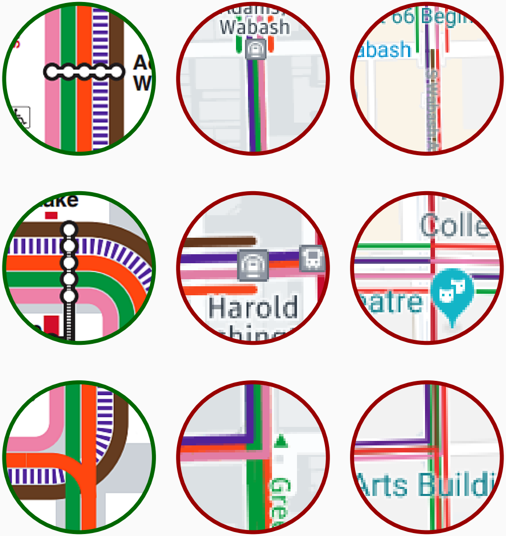


I. Avoid line overlaps

II. Match line orderings

Challenges

Official HERE Google



I. Avoid line overlaps

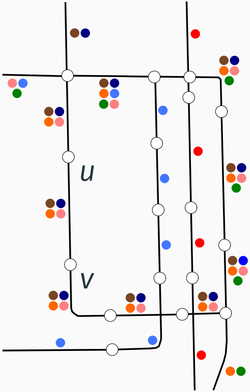
II. Match line orderings

III. Clearly indicate line continuations

Line graph construction

Line graph:

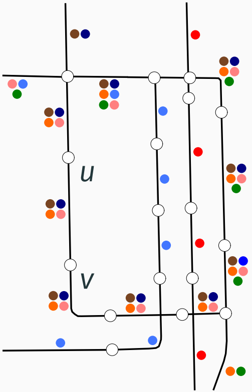
- Undirected labeled graph $G = (V, E, L)$



Line graph construction

Line graph:

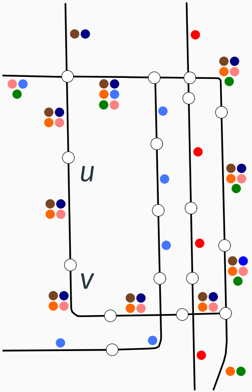
- Undirected labeled graph $G = (V, E, L)$
- Edge labels are subsets of the network lines $\mathcal{L}$ ($L(e) \subseteq \mathcal{L}$)



Line graph construction

Line graph:

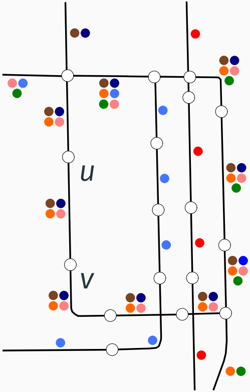
- Undirected labeled graph $G = (V, E, L)$
- Edge labels are subsets of the network lines $\mathcal{L}$ ($L(e) \subseteq \mathcal{L}$)
- Nodes are usually stations



Line graph construction

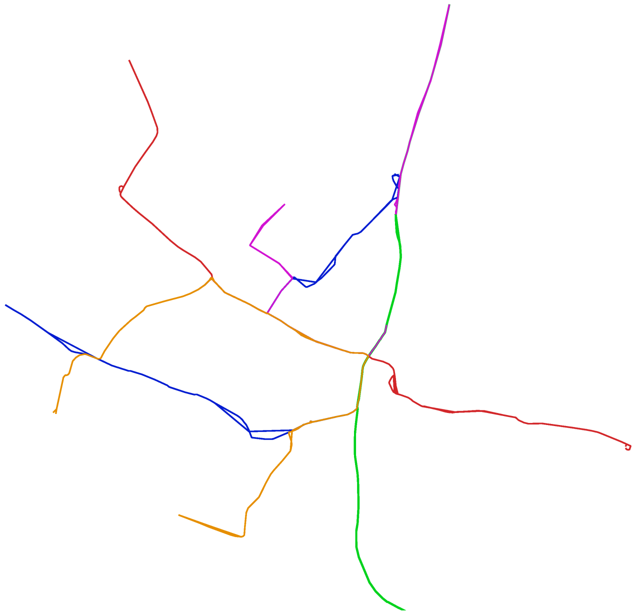
Line graph:

- Undirected labeled graph $G = (V, E, L)$
- Edge labels are subsets of the network lines $\mathcal{L}$ $(L(e) \subseteq \mathcal{L})$
- Nodes are usually stations

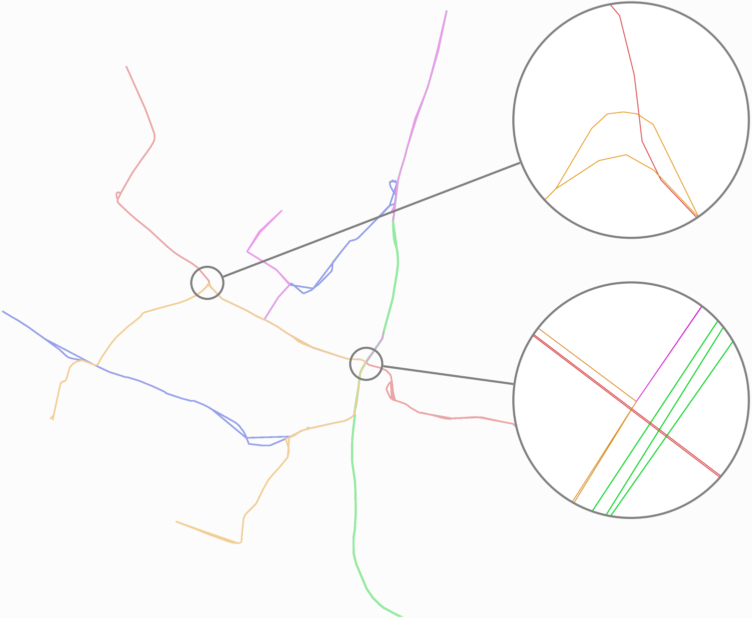


Example: $\mathcal{L} = \{\ \}, L((u, v)) = \{\ \}$

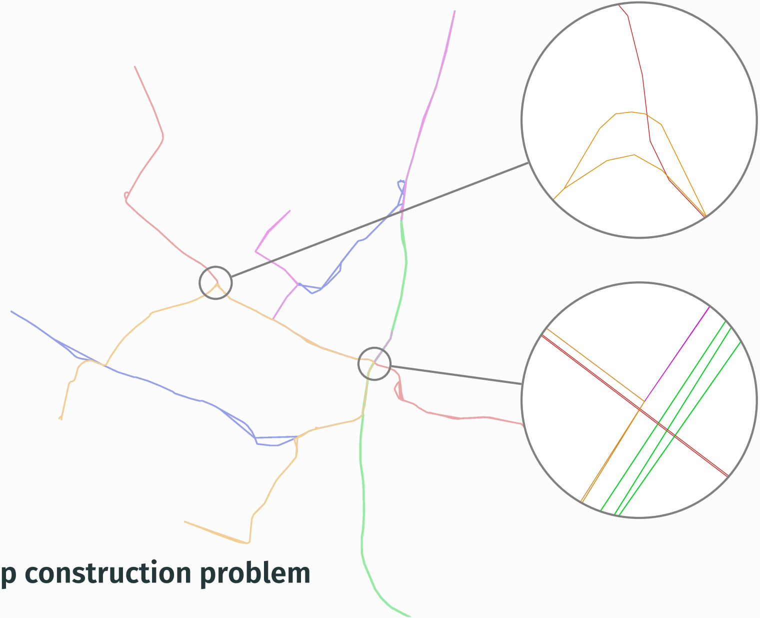
Line graph construction - Input data



Line graph construction - Input data

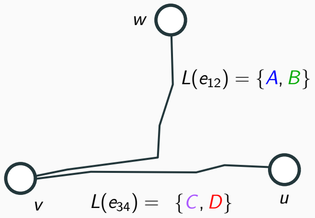


Line graph construction - Input data

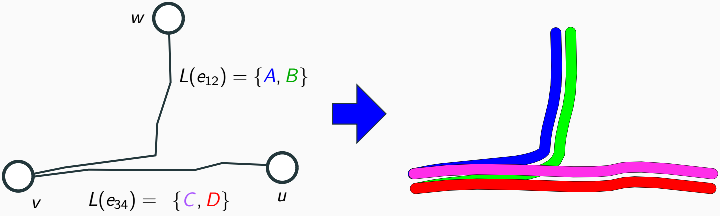


⇒ **Map construction problem**

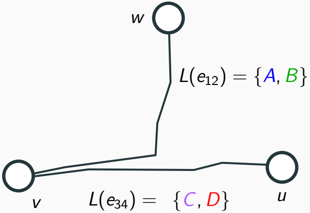
Line graph construction - Non-station nodes

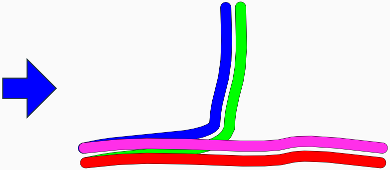


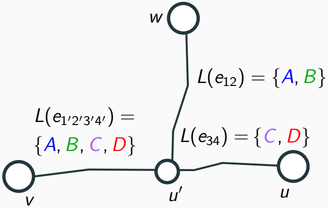
Line graph construction - Non-station nodes



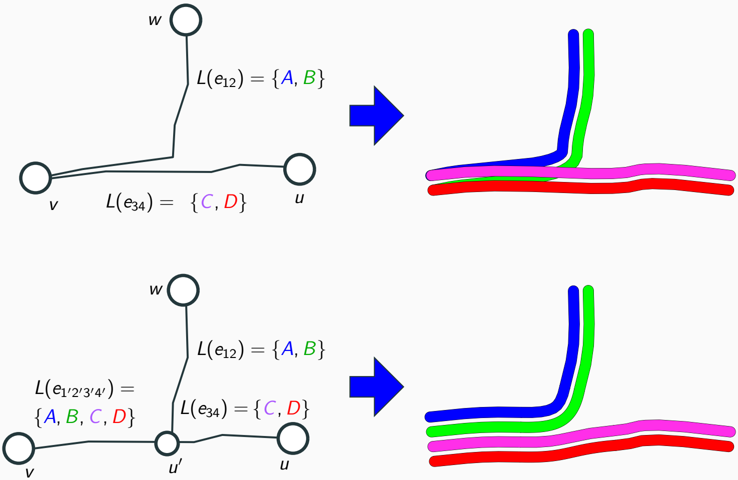
Line graph construction - Non-station nodes



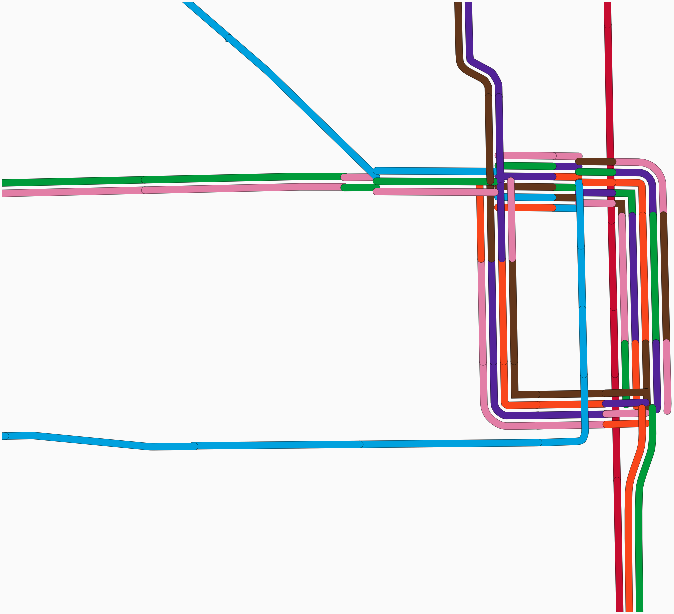




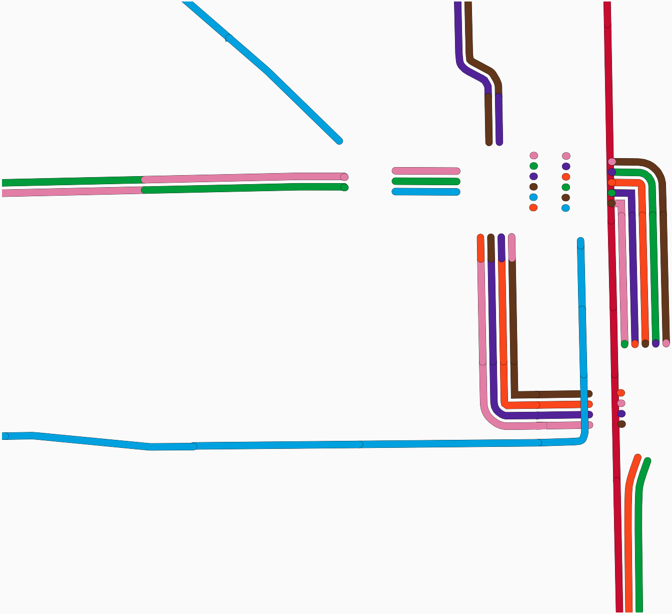
Line graph construction - Non-station nodes



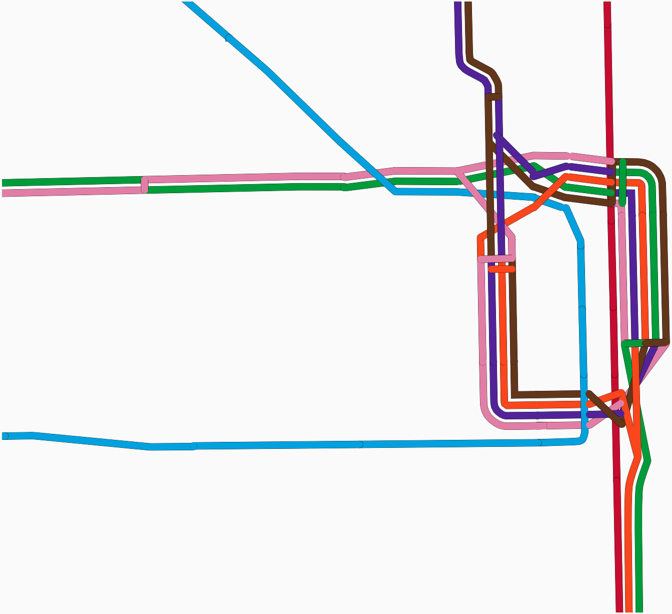
Results so far (1)



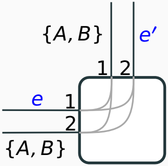
Results so far (1)



Results so far (1)

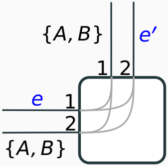


Line-ordering optimization - Baseline ILP



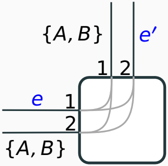
- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$

Line-ordering optimization - Baseline ILP



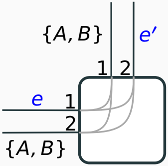
- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$
- Example: x_{eA1} and x_{eA2} for line A

Line-ordering optimization - Baseline ILP



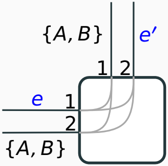
- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$
- Example: x_{eA1} and x_{eA2} for line A
- Constraint: all x_{elp} have to sum up to 1 for a single line l on a single edge e, and for a single p on a single edge

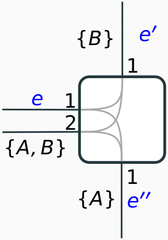
Line-ordering optimization - Baseline ILP



- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$
- Example: x_{eA1} and x_{eA2} for line A
- Constraint: all x_{elp} have to sum up to 1 for a single line l on a single edge e, and for a single p on a single edge
- Standard crossing: Objective variable $x_{ee'AB}$ which is 1 if $p_e(A) < p_e(B)$ and $p_{e'}(A) > p_{e'}(B)$, or else 0

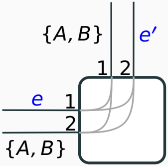
Line-ordering optimization - Baseline ILP

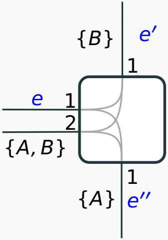




- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$
- Example: x_{eA1} and x_{eA2} for line A
- Constraint: all x_{elp} have to sum up to 1 for a single line l on a single edge e, and for a single p on a single edge
- Standard crossing: Objective variable $x_{ee'AB}$ which is 1 if $p_e(A) < p_e(B)$ and $p_{e'}(A) > p_{e'}(B)$, or else 0
- Split crossing: Objective variable $x_{ee'e''AB}$ which is 1 if $p_e(A) < p_e(B)$, or else 0

Line-ordering optimization - Baseline ILP





- For each edge e, line l and position p, introduce variable $x_{elp} \in 0, 1$
- Example: x_{eA1} and x_{eA2} for line A
- Constraint: all x_{elp} have to sum up to 1 for a single line l on a single edge e, and for a single p on a single edge
- Standard crossing: Objective variable $x_{ee'AB}$ which is 1 if $p_e(A) < p_e(B)$ and $p_{e'}(A) > p_{e'}(B)$, or else 0
- Split crossing: Objective variable $x_{ee'e''AB}$ which is 1 if $p_e(A) < p_e(B)$, or else 0

$\Rightarrow \mathcal{O}(|E|M^2)$ variables, $\mathcal{O}(|E|M^6)$ constraints

Line-ordering optimization - Improved ILP

- **Observation:** we only need to check if $p_e(A) < p_e(B)$ (or vice versa) for both types of crossings

Line-ordering optimization - Improved ILP

- **Observation:** we only need to check if $p_e(A) < p_e(B)$ (or vice versa) for both types of crossings
- But we explicitly enumerate all possible line positions of A and B on e

Line-ordering optimization - Improved ILP

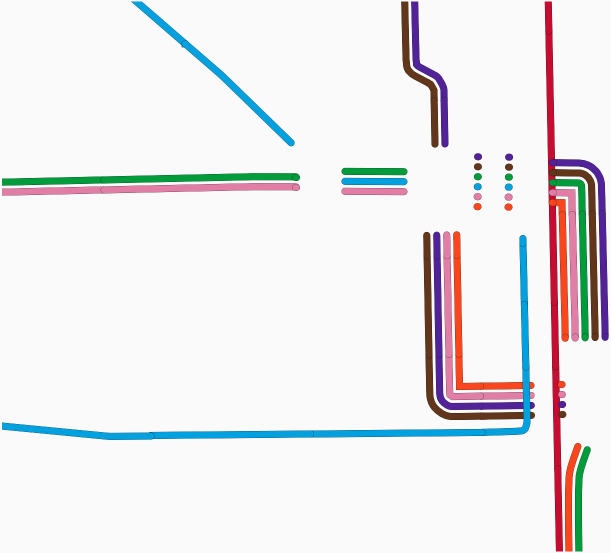
- **Observation:** we only need to check if $p_e(A) < p_e(B)$ (or vice versa) for both types of crossings
- But we explicitly enumerate all possible line positions of A and B on e
- **Basic idea:** introduce binary variables $x_{eA<B}$ and $x_{eB<A}$ which can be efficiently checked

Line-ordering optimization - Improved ILP

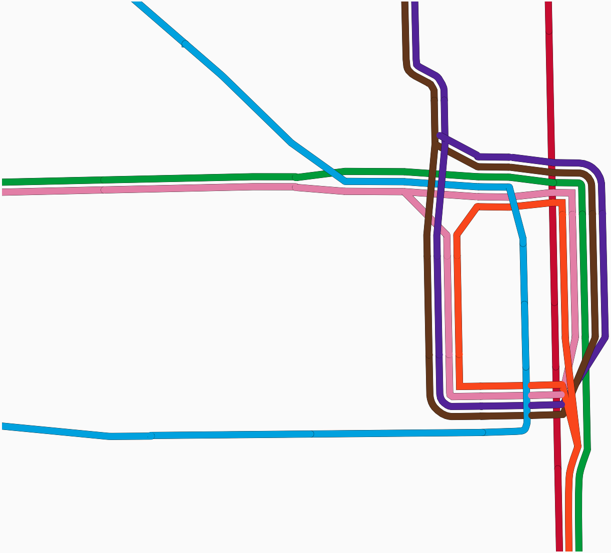
- **Observation:** we only need to check if $p_e(A) < p_e(B)$ (or vice versa) for both types of crossings
- But we explicitly enumerate all possible line positions of A and B on e
- **Basic idea:** introduce binary variables $x_{eA<B}$ and $x_{eB<A}$ which can be efficiently checked

$\Rightarrow \mathcal{O}(|E|M^2)$ variables, $\mathcal{O}(|E|M^2)$ constraints

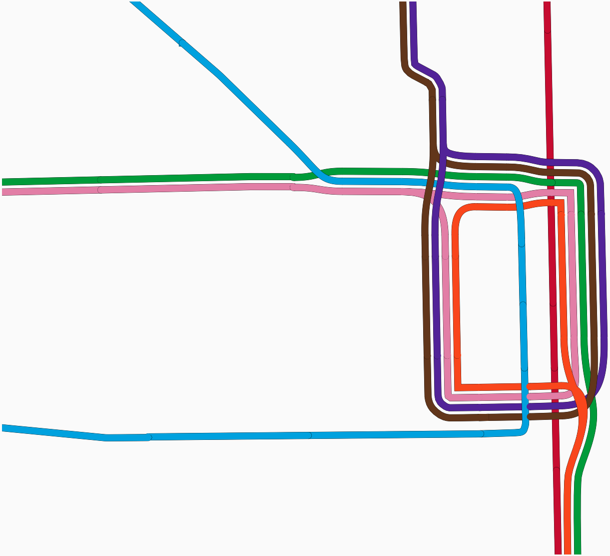
Results so far (2)



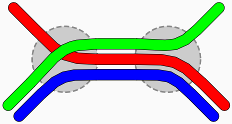
Results so far (2)



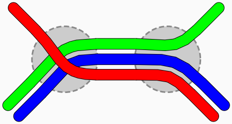
Results so far (2)



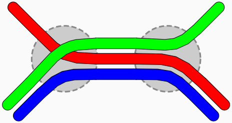
Line-ordering optimization - Line separations



vs.

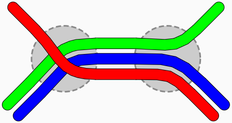


Line-ordering optimization - Line separations



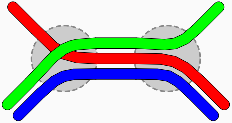
1 crossing

vs.

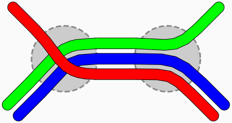


2 crossings

Line-ordering optimization - Line separations

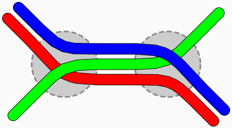


vs.

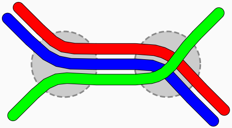


1 crossing

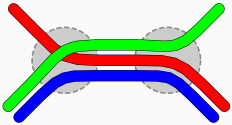
2 crossings



vs.

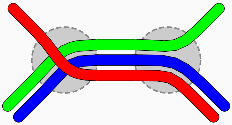


Line-ordering optimization - Line separations

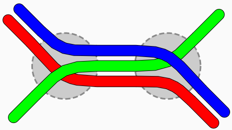


1 crossing

vs.

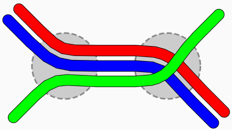


2 crossings



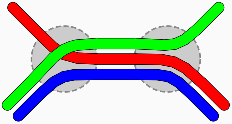
2 crossings

vs.



2 crossings

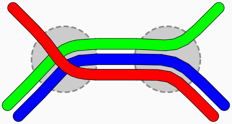
Line-ordering optimization - Line separations



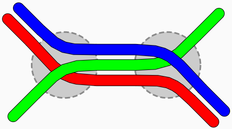
1 crossing

1 separation

vs.



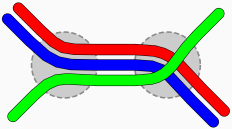
2 crossings



2 crossings

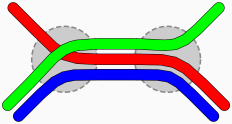
1 separation

vs.



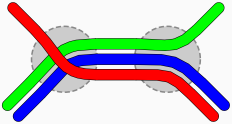
2 crossings

Line-ordering optimization - Line separations

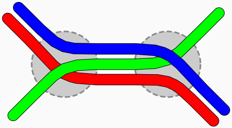


1 crossing
1 separation

vs.

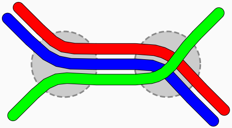


2 crossings
0 separations



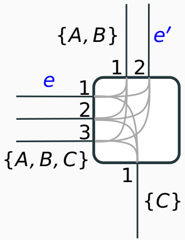
2 crossings
1 separation

vs.



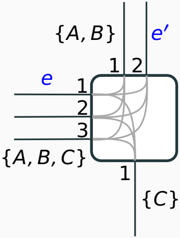
2 crossings
0 separations

Line-ordering optimization - Line separations (ctd.)



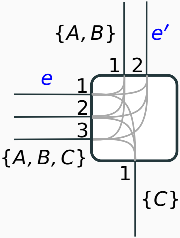
- **Idea:** If two lines A, B continue from e to e', set a binary separation variable $x_{ee'A\|B} = 1$ if they are next to each other in e, but no in e'

Line-ordering optimization - Line separations (ctd.)



- **Idea:** If two lines A, B continue from e to e', set a binary separation variable $x_{ee'A\|B} = 1$ if they are next to each other in e, but no in e'
- Add $x_{ee'A\|B}$ to the objective function

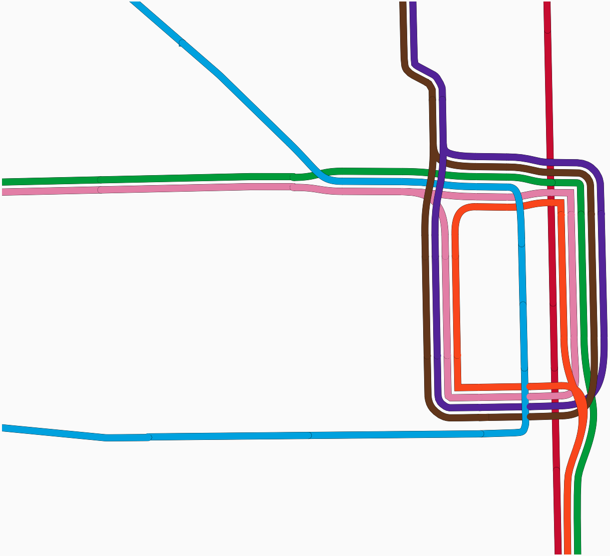
Line-ordering optimization - Line separations (ctd.)



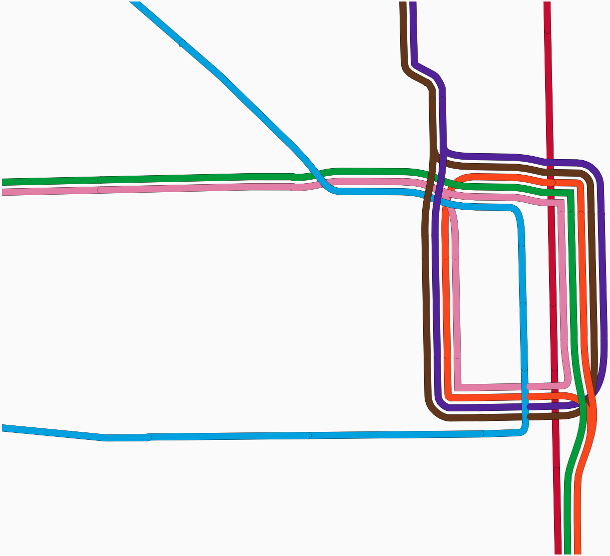
- **Idea:** If two lines A, B continue from e to e', set a binary separation variable $x_{ee'A||B} = 1$ if they are next to each other in e, but no in e'
- Add $x_{ee'A||B}$ to the objective function

$\Rightarrow$ Still $\mathcal{O}(|E|M^2)$ variables, $\mathcal{O}(|E|M^2)$ constraints

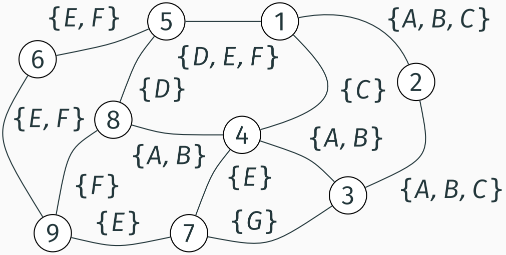
Results so far (3)



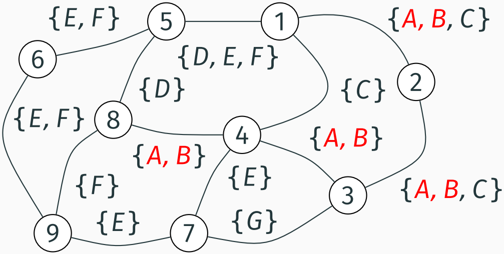
Results so far (3)



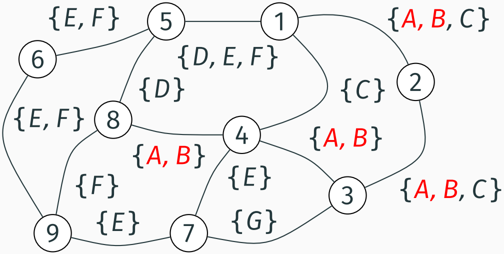
Line-ordering optimization - Core optimization graph



Line-ordering optimization - Core optimization graph

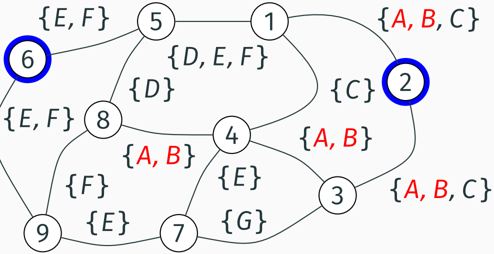


Line-ordering optimization - Core optimization graph



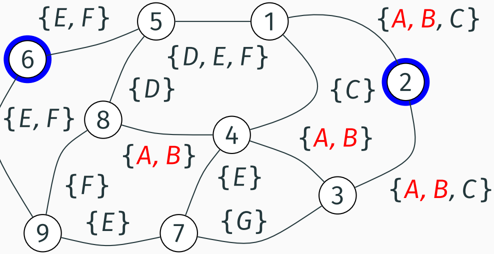
- Combine lines A, B that always occur together into a single new line ($A, B \rightarrow X$)

Line-ordering optimization - Core optimization graph



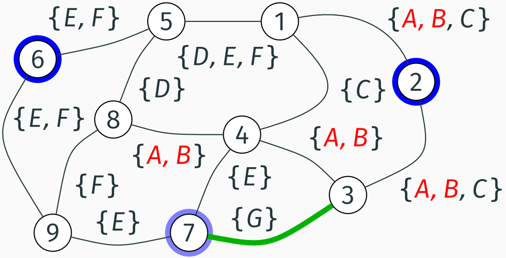
- Combine lines A, B that always occur together into a single new line ($A, B \rightarrow X$)

Line-ordering optimization - Core optimization graph



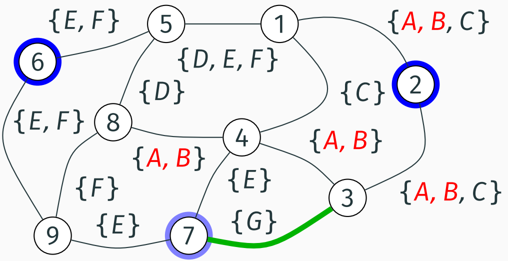
- Combine lines *A*, *B* that always occur together into a single new line ($A, B \longrightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges

Line-ordering optimization - Core optimization graph



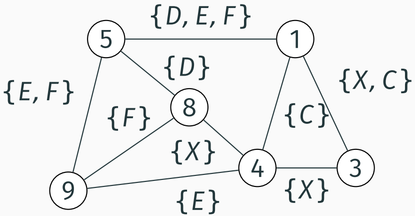
- Combine lines *A*, *B* that always occur together into a single new line ($A, B \to X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges

Line-ordering optimization - Core optimization graph



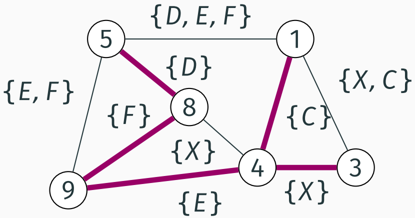
- Combine lines *A*, *B* that always occur together into a single new line ($A, B \rightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges
- Remove edges (u, v) where u and v are termini for all $L((u, v))$

Line-ordering optimization - Core optimization graph



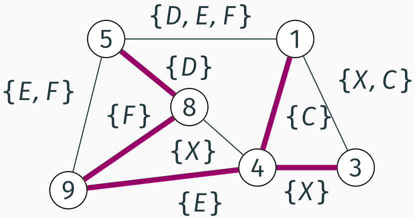
- Combine lines A, B that always occur together into a single new line ($A, B \longrightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges
- Remove edges (u, v) where u and v are termini for all $L((u, v))$

Line-ordering optimization - Core optimization graph



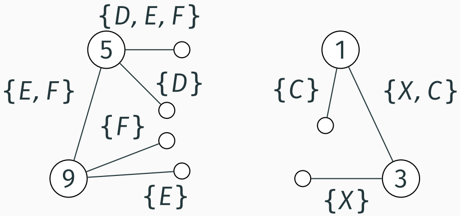
- Combine lines A, B that always occur together into a single new line ($A, B \rightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges
- Remove edges (u, v) where u and v are termini for all $L((u, v))$

Line-ordering optimization - Core optimization graph



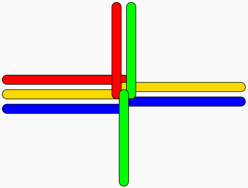
- Combine lines A, B that always occur together into a single new line ($A, B \rightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges
- Remove edges (u, v) where u and v are termini for all $L((u, v))$
- Cut edges with $|L(e)| = 1$

Line-ordering optimization - Core optimization graph



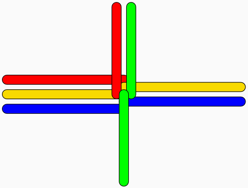
- Combine lines A, B that always occur together into a single new line ($A, B \longrightarrow X$)
- Delete nodes with degree 2 if adjacent edges have the same lines and merge these edges
- Remove edges (u, v) where u and v are termini for all $L((u, v))$
- Cut edges with $|L(e)| = 1$

Rendering

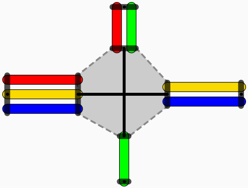


1. Render parallel lines

Rendering

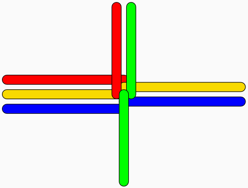


1. Render parallel lines

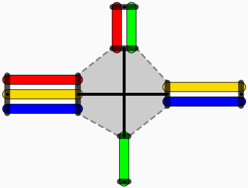


2. Free node space

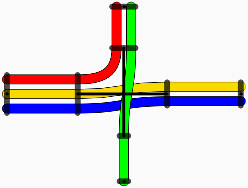
Rendering



1. Render parallel lines

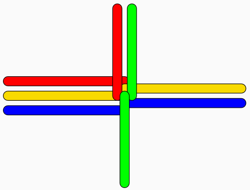


2. Free node space

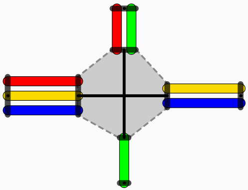


3. Render inner node connections

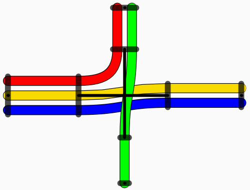
Rendering



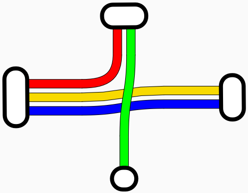
1. Render parallel lines



2. Free node space

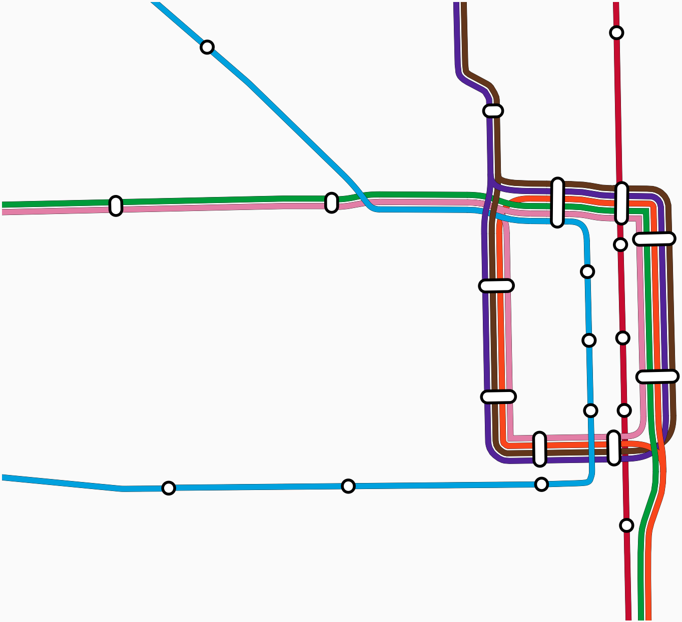


3. Render inner node connections

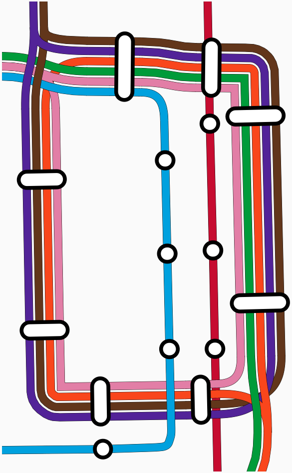


4. Render stations

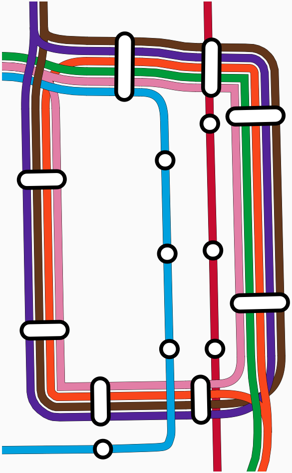
Results so far (4)

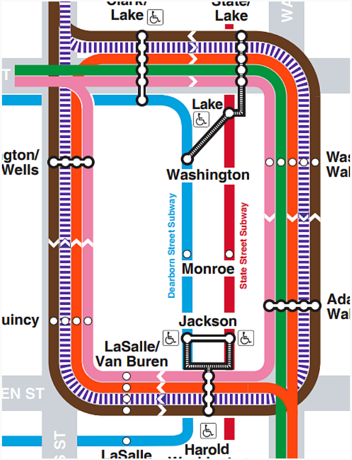


Results so far (4)



Results so far (4)





Evaluation - Line Ordering

T = number of (consecutive) line swaps necessary to transform offical map into our map

	Off. map		Our map		
	×	||	×	||	**T**
Freiburg	7	1	7	0	**2**
Dallas	3	1	3	0	**1**
Chicago	26	0	27	0	**1**
Stuttgart	65	5	64	2	**4**

Evaluation - ILP Solution times

ILP solution times for Chicago, on baseline graph

	rows × cols	GLPK	CBC	GU	×	||
Base	41k × 861	—	—	—	22	4-7
Impr.	1.4k × 982	**9s**	**1s**	**41ms**	22	4-7
+ Sep.	1.9k × 1.2k	**47m**	**19s**	**1.8s**	27	0

Evaluation - ILP Solution times

ILP solution times for Chicago, on baseline graph

	rows × cols	GLPK	CBC	GU	×	||
Base	41k × 861	—	—	—	22	4-7
Impr.	1.4k × 982	**9s**	**1s**	**41ms**	22	4-7
+ Sep.	1.9k × 1.2k	**47m**	**19s**	**1.8s**	27	0

ILP solution times for Chicago, on core graph

	rows × cols	GLPK	CBC	GU	×	||
Base	8.2k × 266	—	**47m**	**2m**	22	4-7
Impr.	394 × 285	**0.8s**	**0.1s**	**10ms**	22	4-7
+ Sep.	505 × 338	**23s**	**3.8s**	**0.3s**	27	0

Future work

- Additional rules for core graph reduction
  (work in progress)

Future work

- Additional rules for core graph reduction
  (work in progress)
- Faster construction times of line graph
  (current state: 1-15 s for our test datasets)

Future work

- Additional rules for core graph reduction (work in progress)
- Faster construction times of line graph (current state: 1-15 s for our test datasets)
- Other sources for input line graph than schedule data (e.g. OSM, work in progress)

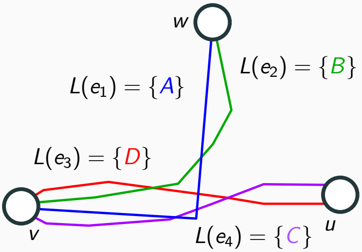
Future work

- Additional rules for core graph reduction (work in progress)
- Faster construction times of line graph (current state: 1-15 s for our test datasets)
- Other sources for input line graph than schedule data (e.g. OSM, work in progress)
- Octilinearize line graph for (non-overlay) schematic metro maps (work in progress)

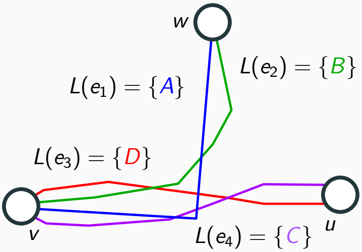
Thank you!

http://loom.informatik.uni-freiburg.de

Line graph construction - Shared segment collapsing

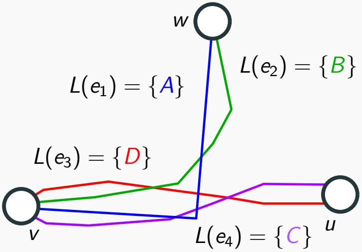


Line graph construction - Shared segment collapsing



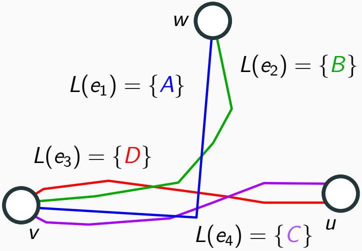
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$

Line graph construction - Shared segment collapsing



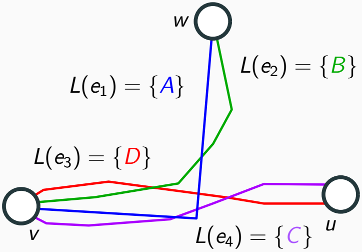
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f

Line graph construction - Shared segment collapsing



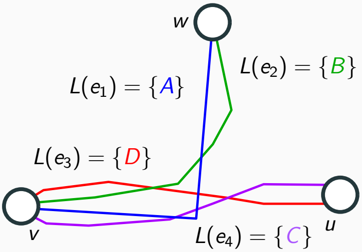
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)

Line graph construction - Shared segment collapsing



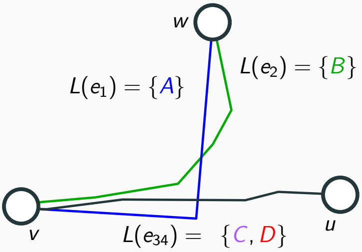
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)
- Take average between the two "shared segments" on e and f

Line graph construction - Shared segment collapsing



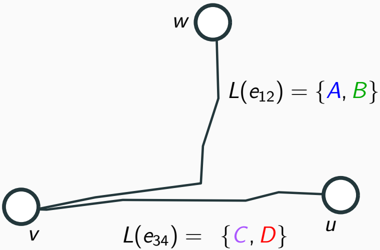
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)
- Take average between the two "shared segments" on e and f
- Add additional non-station nodes at segment boundaries

Line graph construction - Shared segment collapsing



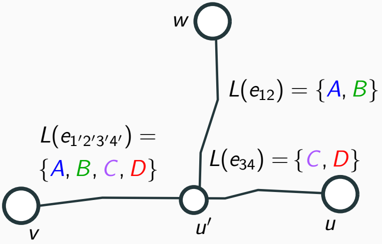
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)
- Take average between the two "shared segments" on e and f
- Add additional non-station nodes at segment boundaries

Line graph construction - Shared segment collapsing



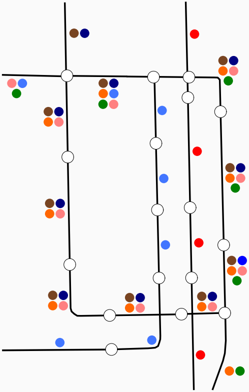
- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)
- Take average between the two "shared segments" on e and f
- Add additional non-station nodes at segment boundaries

Line graph construction - Shared segment collapsing

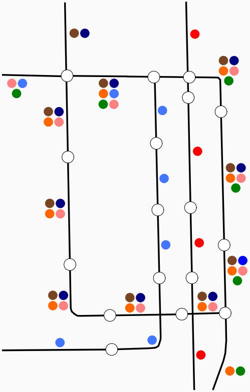


- Repeatedly collapse (segments of) two edges e and f within a distance $\hat{d}$
- Sweep over some edge e in steps of 10 m, measure distance d of current point on e to f
- If $d < \hat{d}$, start new segment. If not, end current (if open)
- Take average between the two "shared segments" on e and f
- Add additional non-station nodes at segment boundaries

Line-ordering optimization - Exhaustive approach

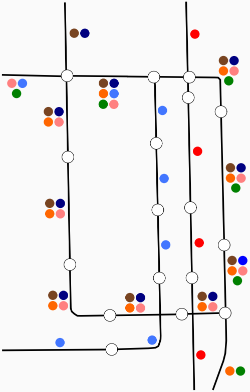


Line-ordering optimization - Exhaustive approach



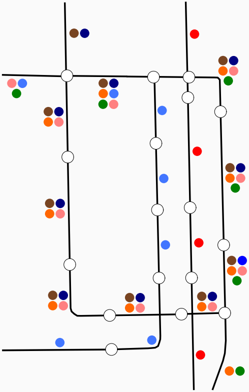
- 23 edges

Line-ordering optimization - Exhaustive approach



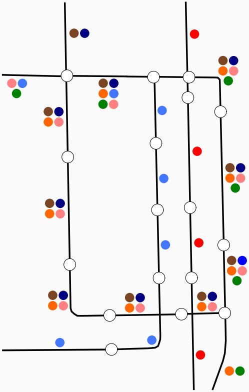
- 23 edges
- Each edge e has $|L(e)|!$ possible line permutations

Line-ordering optimization - Exhaustive approach



- 23 edges
- Each edge e has $|L(e)|!$ possible line permutations
- Possible configurations for the graph on the left: $> 2 \times 10^{17}$

Line-ordering optimization - Exhaustive approach



- 23 edges
- Each edge e has $|L(e)|!$ possible line permutations
- Possible configurations for the graph on the left: $> 2 \times 10^{17}$

$\Rightarrow$ **Naive exhaustive search infeasible**

Baseline ILP - Details

Each line must only be assigned one position:

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

Each position must only be assigned once:

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

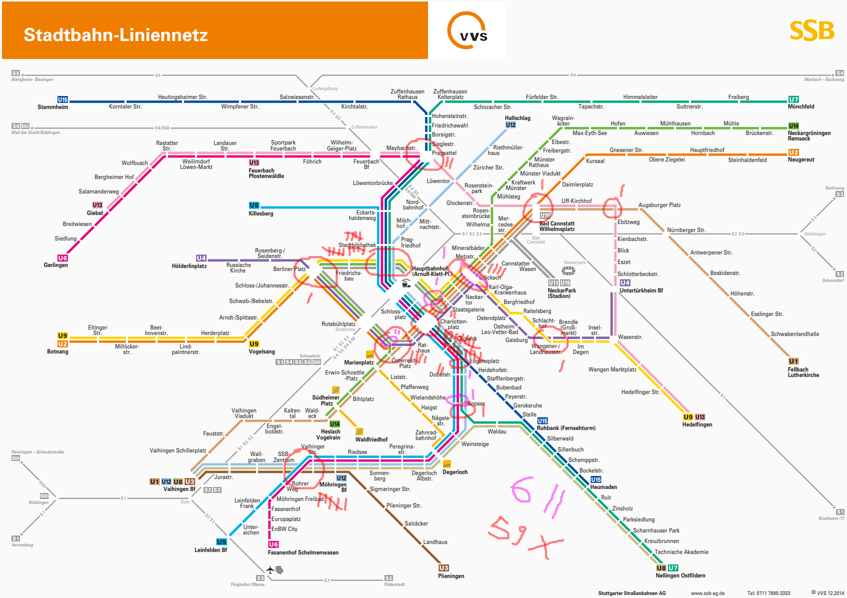
Constraints for ensuring that $x_{ee'AB} = 1$ if a crossing occurs:

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

$$x_{eA2} + x_{eB1} + x_{e'A1} + x_{e'B2} - x_{ee'AB} \leq 3$$

...etc

Stuttgart map - annotated



Dataset dimensions

	t_{extr}	$\lvert\mathcal{S}\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{L}\rvert$	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

Core graph dimensions

	$\|V\|$	$\|E\|$	$\|\mathcal{L}\|$	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