Order in the Underground – How to Automate the Drawing of Metro Maps

Martin Nöllenburg and *Alexander Wolff*

Department of Computer Science

Karlsruhe University

GD 2005

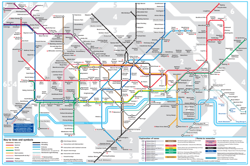
Outline

1. Modeling the Metro Map Problem
   - What is a metro map?
   - Hard and soft constraints
2. Our Solution
   - Mixed-integer programming formulation
   - Experiments
   - Labeling
3. NP-Hardness
   - Rectilinear vs. octilinear drawing
   - Reduction from planar 3-SAT

Outline

1. Modeling the Metro Map Problem
   - What is a metro map?
   - Hard and soft constraints
2. Our Solution
   - Mixed-integer programming formulation
   - Experiments
   - Labeling
3. NP-Hardness
   - Rectilinear vs. octilinear drawing
   - Reduction from planar 3-SAT

What is a Metro Map?



- schematic diagram for public transport
- visualizes lines and stations
- goal: ease navigation for passengers
  - "How do I get from A to B?"
  - "Where to get off and change trains?"
- distorts geometry and scale
- improves readability
- compromise between
  schematic road map $\leftrightarrow$ abstract graph

More Formally

The Metro Map Problem

Given: planar embedded graph $G = (V, E)$, $V \subset \mathbb{R}^2$,
line cover $\mathcal{L}$ of paths or cycles in G (the metro lines),

Goal: draw G and $\mathcal{L}$ nicely.

More Formally

The Metro Map Problem

Given: planar embedded graph $G = (V, E)$, $V \subset \mathbb{R}^2$,
line cover $\mathcal{L}$ of paths or cycles in G (the metro lines),

Goal: draw G and $\mathcal{L}$ nicely.

- What is a nice drawing?

More Formally

The Metro Map Problem

Given: planar embedded graph $G = (V, E)$, $V \subset \mathbb{R}^2$,
line cover $\mathcal{L}$ of paths or cycles in G (the metro lines),

Goal: draw G and $\mathcal{L}$ nicely.

- What is a nice drawing?
- Look at real-world metro maps drawn by graphic designers and model their design principles as

More Formally

The Metro Map Problem

Given: planar embedded graph $G = (V, E)$, $V \subset \mathbb{R}^2$,
line cover $\mathcal{L}$ of paths or cycles in G (the metro lines),

Goal: draw G and $\mathcal{L}$ nicely.

- What is a nice drawing?
- Look at real-world metro maps drawn by graphic designers and model their design principles as
  - *hard* constraints – must be fulfilled,

More Formally

The Metro Map Problem

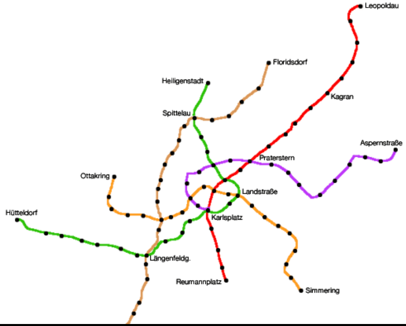
Given: planar embedded graph $G = (V, E)$, $V \subset \mathbb{R}^2$,
line cover $\mathcal{L}$ of paths or cycles in G (the metro lines),

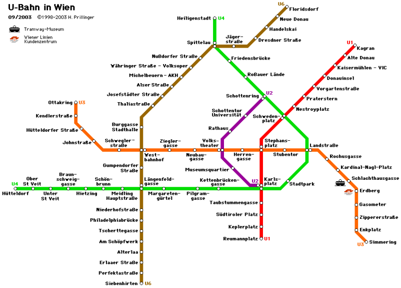
Goal: draw G and $\mathcal{L}$ nicely.

- What is a nice drawing?
- Look at real-world metro maps drawn by graphic designers and model their design principles as
  - *hard* constraints – must be fulfilled,
  - *soft* constraints – should hold as tightly as possible.

Hard Constraints

(H1) preserve embedding of G

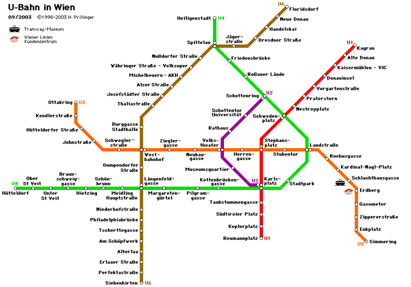




Hard Constraints

(H1) preserve embedding of G

(H2) draw all edges as octilinear line segments,
i.e., parallel to a coordinate axes or at 45° degrees



Hard Constraints

(H1) preserve embedding of G

(H2) draw all edges as octilinear line segments, i.e., parallel to a coordinate axes or at 45° degrees

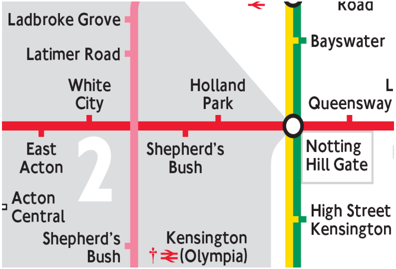
(H3) draw each edge e with length $\geq \ell_e$

Hard Constraints

- (H1) preserve embedding of G
- (H2) draw all edges as octilinear line segments, i.e., parallel to a coordinate axes or at 45° degrees
- (H3) draw each edge e with length $\geq \ell_e$
- (H4) keep vertices d_{min} away from non-incident edges

Soft Constraints

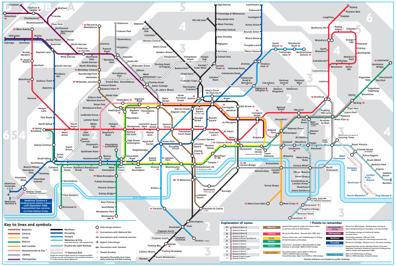
(S1) draw metro lines with few bends



Soft Constraints

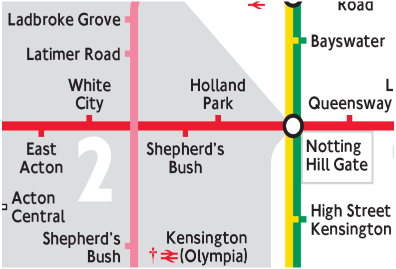
(S1) draw metro lines with few bends

(S2) keep total edge length small



Soft Constraints

- (S1) draw metro lines with few bends
- (S2) keep total edge length small
- (S3) draw each octilinear edge similar to its geographical orientation: keep its relative position



Outline

1. Modeling the Metro Map Problem
   - What is a metro map?
   - Hard and soft constraints
2. Our Solution
   - Mixed-integer programming formulation
   - Experiments
   - Labeling
3. NP-Hardness
   - Rectilinear vs. octilinear drawing
   - Reduction from planar 3-SAT

Mathematical Programming

- Linear Programming: efficient optimization method for
  - linear constraints and objective function,
  - real-valued variables (domain $\mathbb{R}$).

Mathematical Programming

- Linear Programming: efficient optimization method for
  - linear constraints and objective function,
  - real-valued variables (domain $\mathbb{R}$).
- Mixed-Integer Programming (MIP):
  - allows also integer variables (domain $\mathbb{Z}$),
  - solution NP-hard in general.

Mathematical Programming

- Linear Programming: efficient optimization method for
  - linear constraints and objective function,
  - real-valued variables (domain $\mathbb{R}$).
- Mixed-Integer Programming (MIP):
  - allows also integer variables (domain $\mathbb{Z}$),
  - solution NP-hard in general.
- Still a practical method for many hard optimizat. problems.

Mathematical Programming

- Linear Programming: efficient optimization method for
  - linear constraints and objective function,
  - real-valued variables (domain $\mathbb{R}$).
- Mixed-Integer Programming (MIP):
  - allows also integer variables (domain $\mathbb{Z}$),
  - solution NP-hard in general.
- Still a practical method for many hard optimizat. problems.

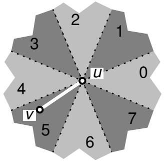
Theorem (Nöllenburg & Wolff GD'05)

The problem MetroMapLayout can be formulated as a MIP s.th.

linear constraints $\rightarrow$ hard constraints,
objective function $\rightarrow$ soft constraints.

Example: Octilinearity and Relative Position

Example: Octilinearity and Relative Position

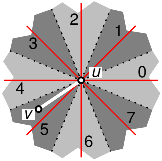


Sectors

– for each vtx. u partition plane into sectors 0–7

- here: $\mathrm{sec}(u, v) = 5$ (input)

Example: Octilinearity and Relative Position



Sectors

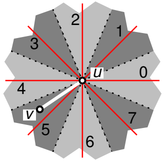
– for each vtx. u partition plane into sectors 0–7

- here: $\mathrm{sec}(u, v) = 5$ (input)

– number octilinear edge directions accordingly

- e.g., $\mathrm{dir}(u, v) = 4$ (output)

Example: Octilinearity and Relative Position



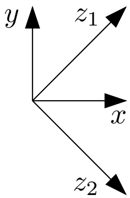
Sectors

– for each vtx. u partition plane into sectors 0–7

- here: $\text{sec}(u, v) = 5$ (input)

– number octilinear edge directions accordingly

- e.g., $\text{dir}(u, v) = 4$ (output)



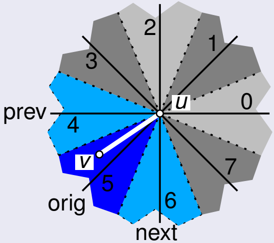
Coordinates

assign z_1- and z_2-coordinates to each vertex v:

- $z_1(v) = x(v) + y(v)$
- $z_2(v) = x(v) - y(v)$

Example: Octilinearity and Relative Position

Goal

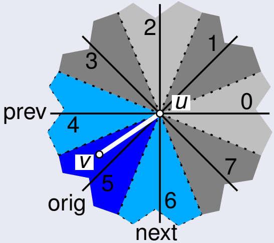


Draw edge uv

- with minimum length ℓ_{uv}
- restricted to 3 directions

Example: Octilinearity and Relative Position

Goal



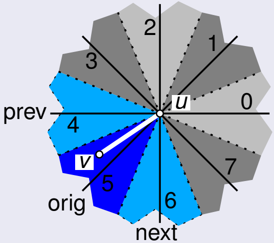
Draw edge *uv*

- with minimum length ℓ_{uv}
- restricted to 3 directions

How to model this using linear constraints?

Example: Octilinearity and Relative Position

Goal



Draw edge uv

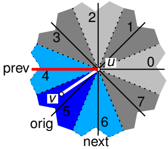
- with minimum length ℓ_{uv}
- restricted to 3 directions

How to model this using linear constraints?

Binary Variables

$$\alpha_{\text{prev}}(u, v) + \alpha_{\text{orig}}(u, v) + \alpha_{\text{next}}(u, v) = 1$$

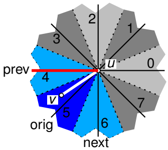
Example: Octilinearity and Relative Position



Previous Sector

$$\begin{aligned} y(u) - y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\ -y(u) + y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\ x(u) - x(v) &\geq -M(1 - \alpha_{\text{prev}}(u, v)) + \ell_{uv} \end{aligned}$$

Example: Octilinearity and Relative Position

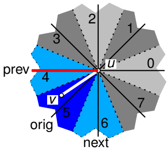


Previous Sector

$$
\begin{aligned}
y(u) - y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\
-y(u) + y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\
x(u) - x(v) &\geq -M(1 - \alpha_{\text{prev}}(u, v)) + \ell_{uv}
\end{aligned}
$$

How does this work?

Example: Octilinearity and Relative Position



Previous Sector

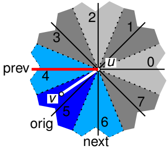
$$
\begin{aligned}
y(u) - y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\
-y(u) + y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\
x(u) - x(v) &\geq -M(1 - \alpha_{\text{prev}}(u, v)) + \ell_{uv}
\end{aligned}
$$

How does this work?

Case 1: $\alpha_{\text{prev}}(u, v) = 0$

$$
\begin{aligned}
y(u) - y(v) &\leq M \\
-y(u) + y(v) &\leq M \\
x(u) - x(v) &\geq \ell_{uv} - M
\end{aligned}
$$

Example: Octilinearity and Relative Position



Previous Sector

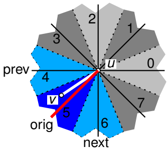
$$\begin{aligned} y(u) - y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\ -y(u) + y(v) &\leq M(1 - \alpha_{\text{prev}}(u, v)) \\ x(u) - x(v) &\geq -M(1 - \alpha_{\text{prev}}(u, v)) + \ell_{uv} \end{aligned}$$

How does this work?

Case 2: $\alpha_{\text{prev}}(u, v) = 1$

$$\begin{aligned} y(u) - y(v) &\leq 0 \\ -y(u) + y(v) &\leq 0 \\ x(u) - x(v) &\geq \ell_{uv} \end{aligned}$$

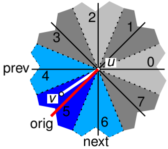
Example: Octilinearity and Relative Position



Original Sector

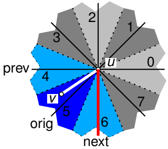
$$\begin{aligned} z_2(u) - z_2(v) &\leq M(1 - \alpha_{\mathrm{orig}}(u,v)) \\ -z_2(u) + z_2(v) &\leq M(1 - \alpha_{\mathrm{orig}}(u,v)) \\ z_1(u) - z_1(v) &\geq -M(1 - \alpha_{\mathrm{orig}}(u,v)) + 2\ell_{uv} \end{aligned}$$

Example: Octilinearity and Relative Position



Original Sector

$$
\begin{aligned}
z_2(u) - z_2(v) &\leq M(1 - \alpha_{\mathrm{orig}}(u, v)) \\
-z_2(u) + z_2(v) &\leq M(1 - \alpha_{\mathrm{orig}}(u, v)) \\
z_1(u) - z_1(v) &\geq -M(1 - \alpha_{\mathrm{orig}}(u, v)) + 2\ell_{uv}
\end{aligned}
$$



Next Sector

$$
\begin{aligned}
x(u) - x(v) &\leq M(1 - \alpha_{\mathrm{next}}(u, v)) \\
-x(u) + x(v) &\leq M(1 - \alpha_{\mathrm{next}}(u, v)) \\
y(u) - y(v) &\geq -M(1 - \alpha_{\mathrm{next}}(u, v)) + \ell_{uv}
\end{aligned}
$$

Summary of the Model

- The above constraints enforce
  - octilinearity,
  - minimum edge length,
  - (partially) relative position

Summary of the Model

- The above constraints enforce
  - octilinearity,
  - minimum edge length,
  - (partially) relative position
- Similarly:
  - preservation of embedding
  - planarity

Summary of the Model

- The above constraints enforce
  - octilinearity,
  - minimum edge length,
  - (partially) relative position
- Similarly:
  - preservation of embedding
  - planarity
- Soft constraints: weighted sum in objective function

$$\text{minimize } \lambda_{\text{bends}} \, \text{cost}_{\text{bends}} + \lambda_{\text{length}} \, \text{cost}_{\text{length}} + \lambda_{\text{dir}} \, \text{cost}_{\text{dir}}$$

Summary of the Model

- The above constraints enforce
  - octilinearity,
  - minimum edge length,
  - (partially) relative position
- Similarly:
  - preservation of embedding
  - planarity
- Soft constraints: weighted sum in objective function

$$\text{minimize } \lambda_{\text{bends}}\,\text{cost}_{\text{bends}} + \lambda_{\text{length}}\,\text{cost}_{\text{length}} + \lambda_{\text{dir}}\,\text{cost}_{\text{dir}}$$

- In total $O(|V|^2)$ constraints and variables

Summary of the Model

- The above constraints enforce
  - octilinearity,
  - minimum edge length,
  - (partially) relative position
- Similarly:
  - preservation of embedding
  - planarity
- Soft constraints: weighted sum in objective function

$$\text{minimize } \lambda_{\text{bends}} \, \text{cost}_{\text{bends}} + \lambda_{\text{length}} \, \text{cost}_{\text{length}} + \lambda_{\text{dir}} \, \text{cost}_{\text{dir}}$$

- In total $O(|V|^2)$ constraints and variables

Results – Vienna

Input

Input	$\|V\|$	$\|E\|$	lines
normal	90	96	5
reduced	44	50	

Results – Vienna

Input

Input	$\|V\|$	$\|E\|$	lines
normal	90	96	5
reduced	44	50	

$\downarrow$

MIP	constr.	var.
normal	39363	9960
improved	23226	6048
heuristic 1	5703	1800
heuristic 2	1875	872

Results – Vienna

Input

Input	$\|V\|$	$\|E\|$	lines
normal	90	96	5
reduced	44	50	

↓

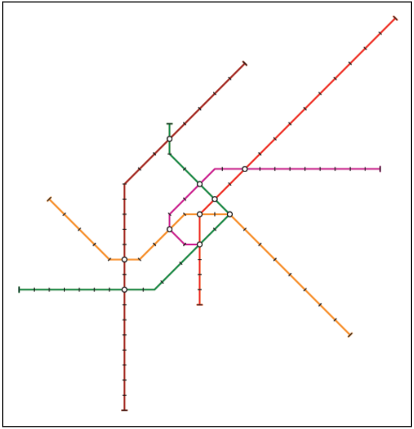
MIP	constr.	var.
normal	39363	9960
improved	23226	6048
heuristic 1	5703	1800
heuristic 2*	1875	872

*) 29 seconds w/o proof of opt.

Results – Vienna

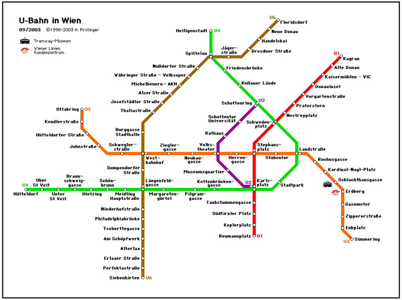
Input

Output

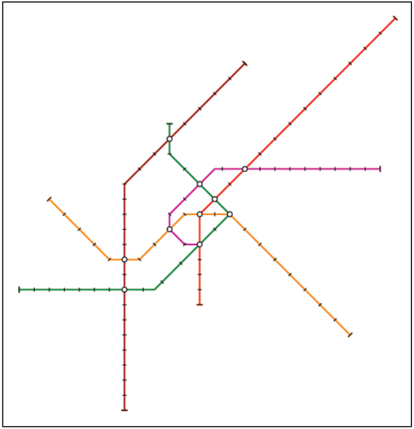


Results – Vienna

Official map

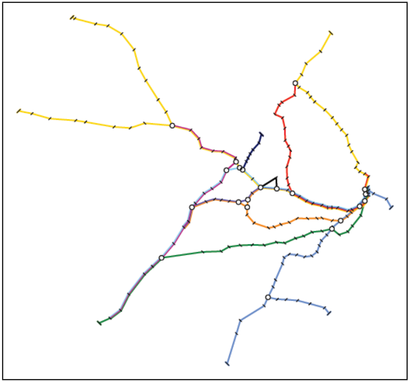


Output



Results – Sydney

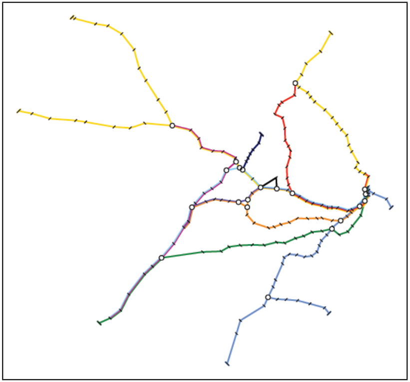
Input



Input	$\|V\|$	$\|E\|$	lines
normal	174	183	10
reduced	62	71	

Results – Sydney

Input



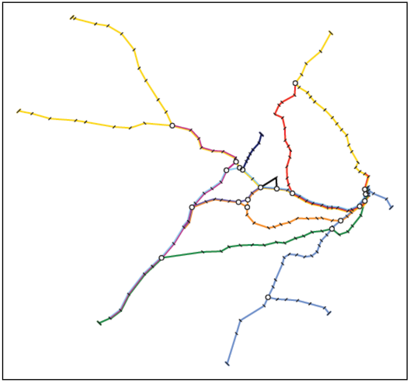
Input	$\|V\|$	$\|E\|$	lines
normal	174	183	10
reduced	62	71	

$\downarrow$

MIP	constr.	var.
normal	81416	20329
improved	45182	11545
heuristic 1	6242	2105
heuristic 2	3041	1329

Results – Sydney

Input



Input	$\lvert V \rvert$	$\lvert E \rvert$	lines
normal	174	183	10
reduced	62	71	

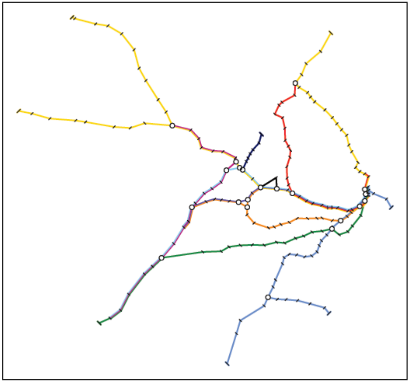
$\downarrow$

MIP	constr.	var.
normal	81416	20329
improved	45182	11545
heuristic 1*	6242	2105
heuristic 2	3041	1329

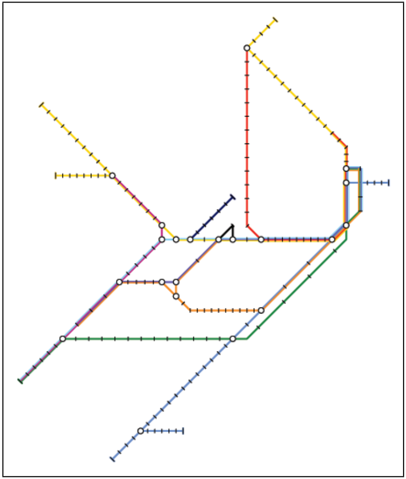
*) 22 minutes w/o proof of opt.

Results – Sydney

Input

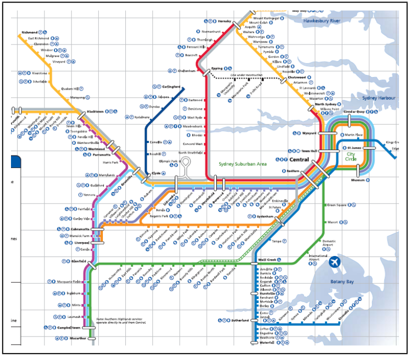


Output

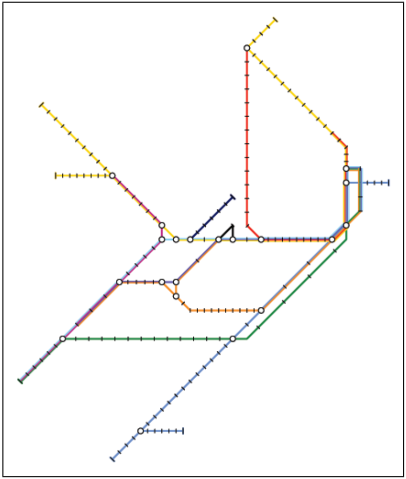


Results – Sydney

Official map

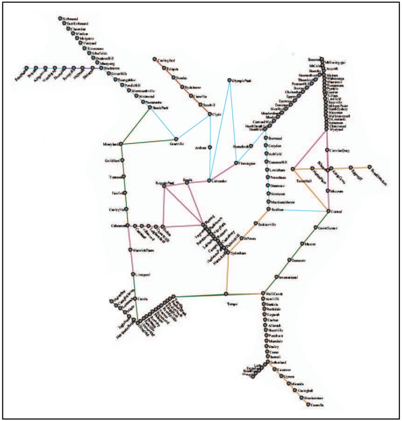


Output

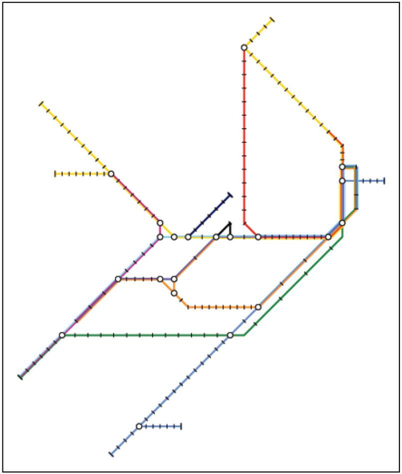


Sydney: Related Work

[Hong et al.: GD04] 7.6 sec.

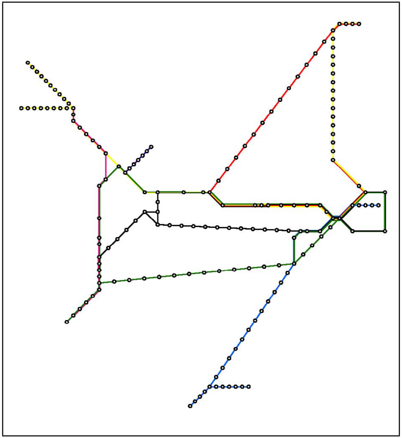


Output (22 min.)

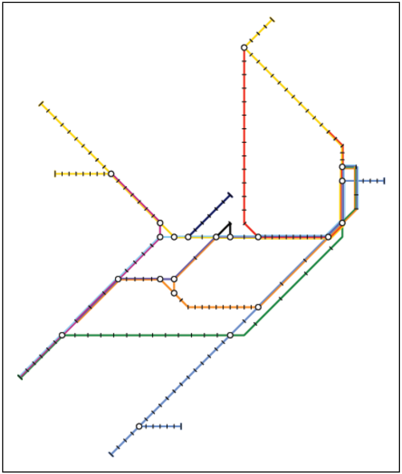


Sydney: Related Work

[Stott & Rodgers: IV04] reduced: 4 min.

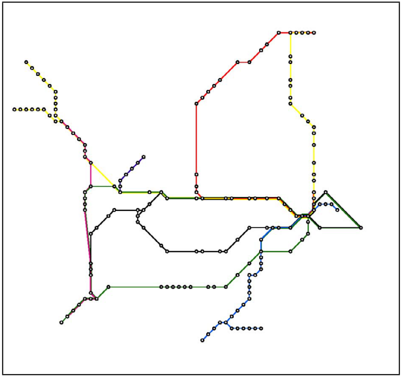


Output (22 min.)

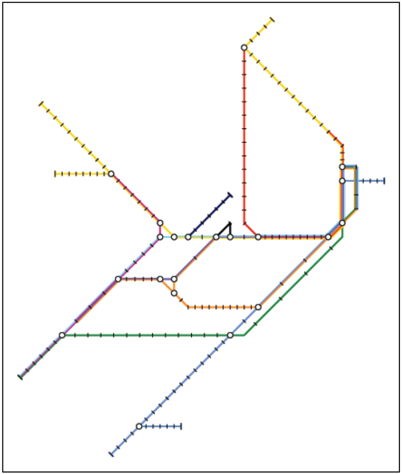


Sydney: Related Work

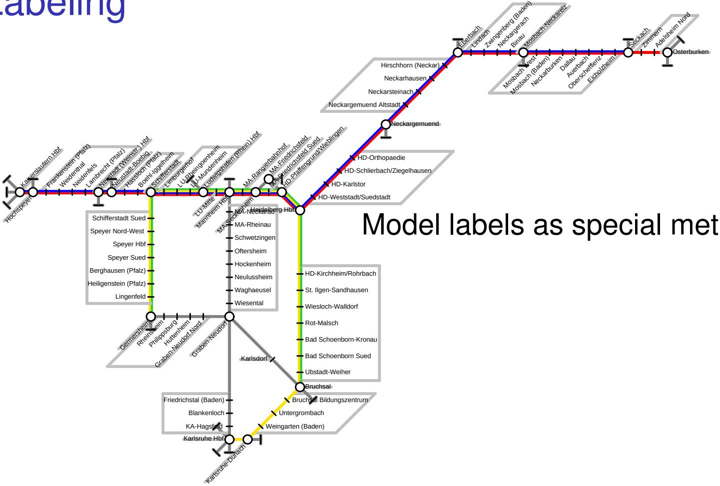
[Stott & Rodgers: IV04] normal: 28 min.



Output (22 min.)

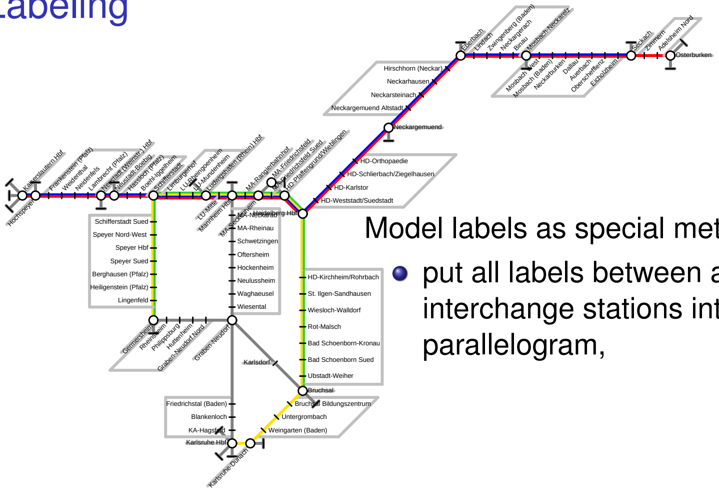


Labeling



Model labels as special metro lines:

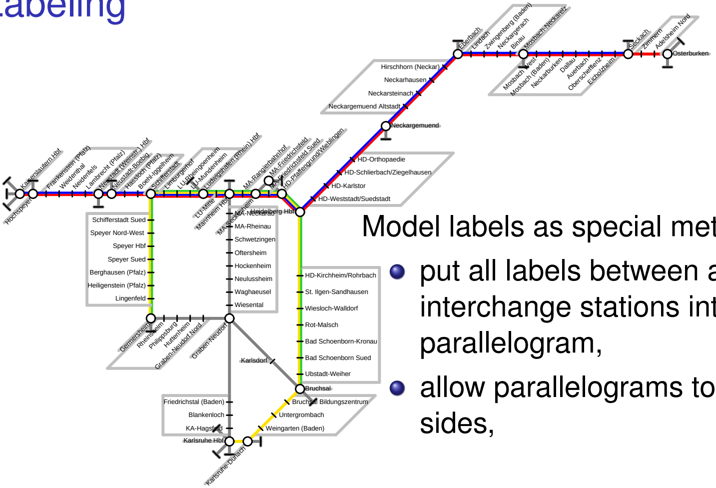
Labeling



Model labels as special metro lines:

- put all labels between a pair of interchange stations into one parallelogram,

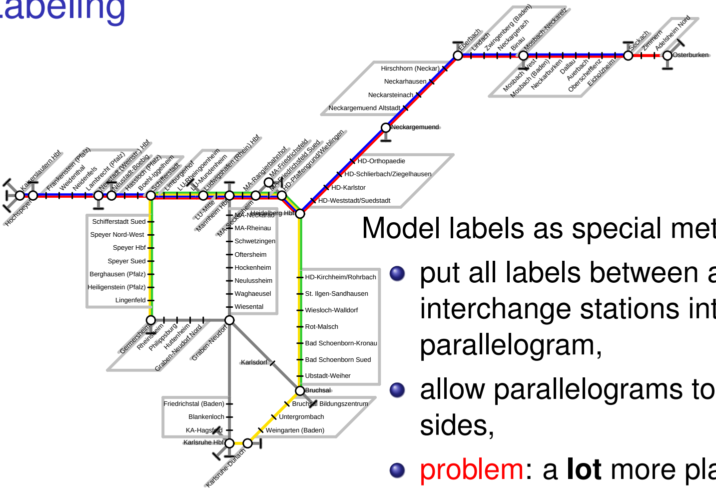
Labeling



Model labels as special metro lines:

- put all labels between a pair of interchange stations into one parallelogram,
- allow parallelograms to change sides,

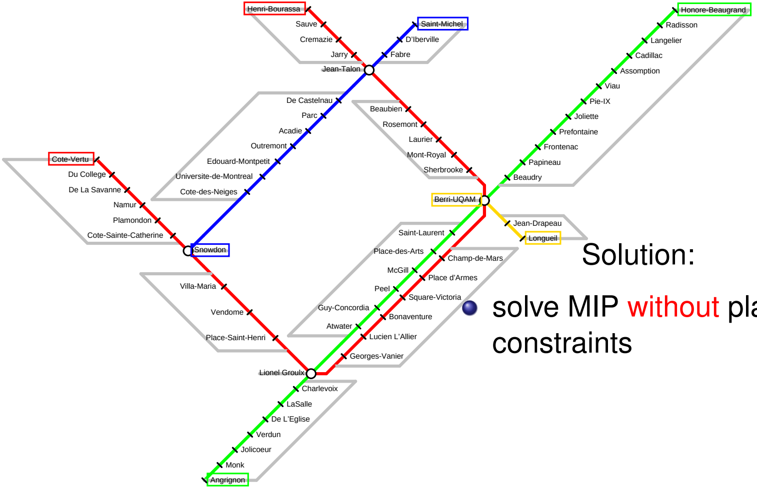
Labeling



Model labels as special metro lines:

- put all labels between a pair of interchange stations into one parallelogram,
- allow parallelograms to change sides,
- problem: a **lot** more planarity constraints :-(

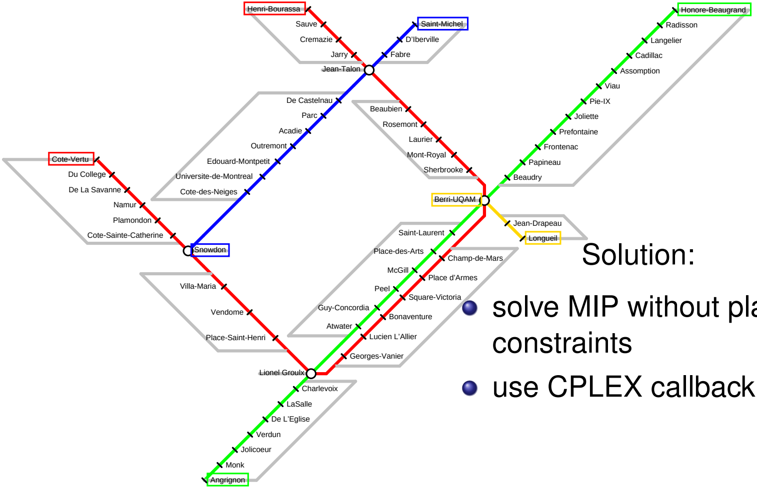
Labeling



Solution:

- solve MIP without planarity constraints

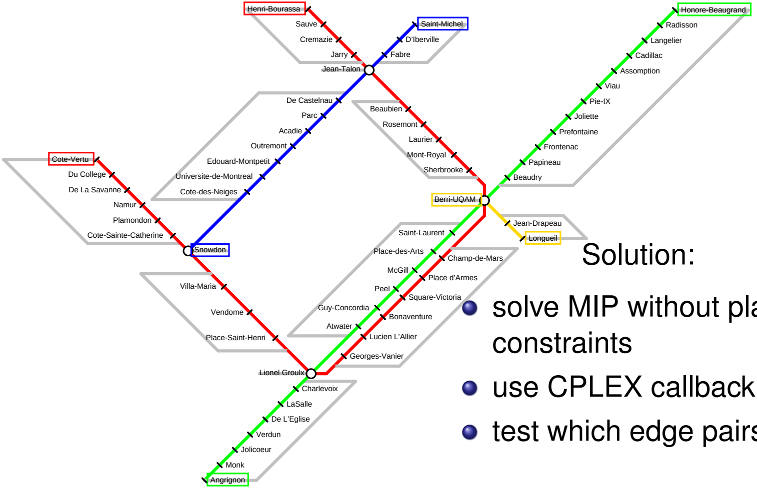
Labeling



Solution:

- solve MIP without planarity constraints
- use CPLEX callback fct.

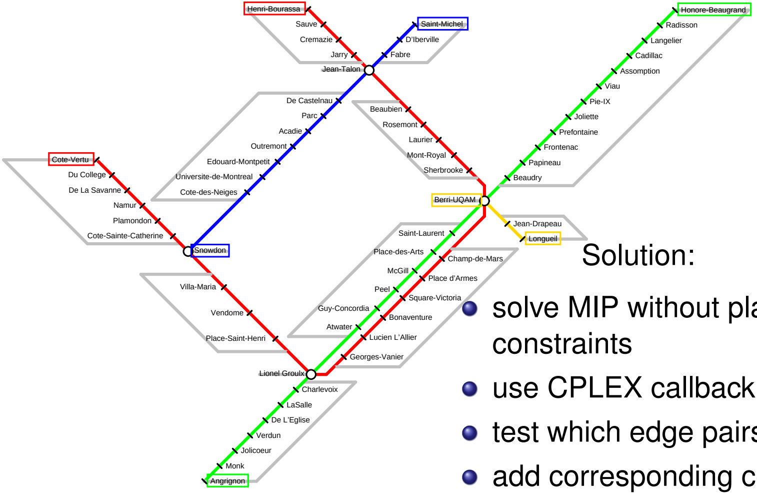
Labeling



Solution:

- solve MIP without planarity constraints
- use CPLEX callback fct.
- test which edge pairs cross

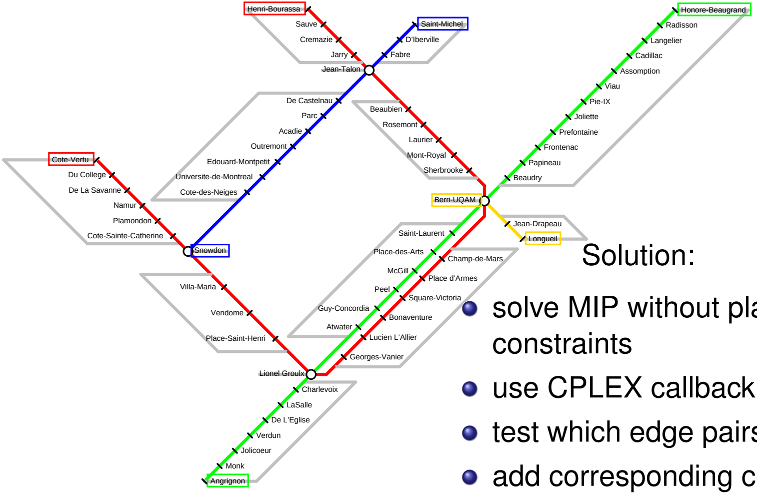
Labeling



Solution:

- solve MIP without planarity constraints
- use CPLEX callback fct.
- test which edge pairs cross
- add corresponding constr.

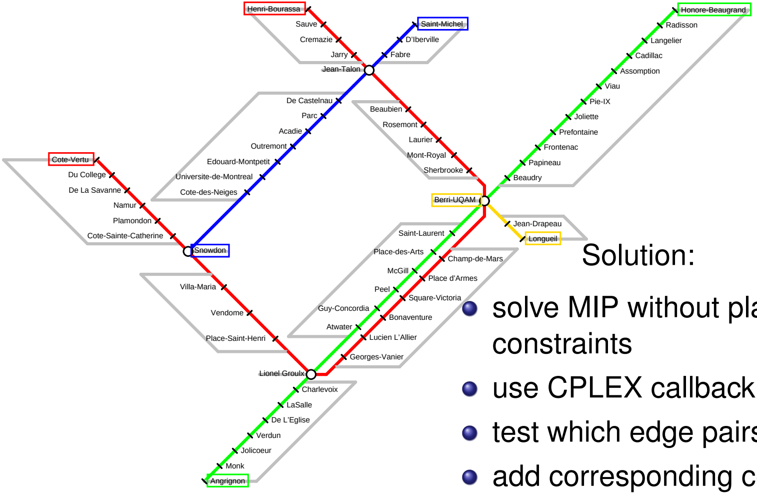
Labeling



Solution:

- solve MIP without planarity constraints
- use CPLEX callback fct.
- test which edge pairs cross
- add corresponding constr.
- continue to solve MIP

Labeling



Montréal: 17 min.

Solution:

- solve MIP without planarity constraints
- use CPLEX callback fct.
- test which edge pairs cross
- add corresponding constr.
- continue to solve MIP

Labeling

Wien: 1 day.

Labeling

Tokyo: $<$ 10 sec.

[Kameda & Imai: IEICE’03]

Outline

Another Problem

RectilinearGraphDrawing Decision Problem

Given a planar embedded graph G with max degree 4.
Is there a drawing of G that

- preserves the embedding,
- uses straight-line edges,
- is rectilinear?

Another Problem

RectilinearGraphDrawing Decision Problem

Given a planar embedded graph G with max degree 4.
Is there a drawing of G that

- preserves the embedding,
- uses straight-line edges,
- is rectilinear?

Theorem (Tamassia '87)

RectilinearGraphDrawing *can be solved efficiently.*

Another Problem

RectilinearGraphDrawing Decision Problem

Given a planar embedded graph G with max degree 4.
Is there a drawing of G that

- preserves the embedding,
- uses straight-line edges,
- is rectilinear?

Theorem (Tamassia '87)

RectilinearGraphDrawing *can be solved efficiently.*

Proof.

By reduction to a flow problem. □

Another Problem

RECTILINEARGRAPHDRAWING Decision Problem

Given a planar embedded graph G with max degree 4.
Is there a drawing of G that

- preserves the embedding,
- uses straight-line edges,
- is rectilinear?

Theorem (Tamassia '87)

RECTILINEARGRAPHDRAWING *can be solved efficiently.*

Proof.

By reduction to a flow problem. □

Our Problem

MetroMapLayout Decision Problem

Given a planar embedded graph G with max degree 8.
Is there a drawing of G that

- preserves the embedding,
- uses straight-line edges,
- is octilinear?

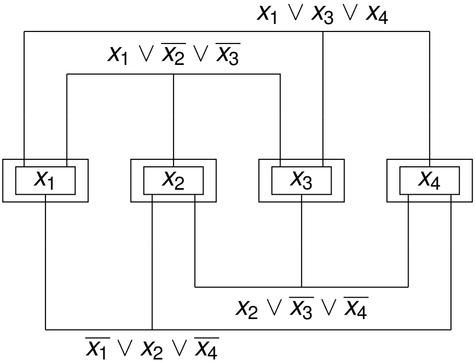
Theorem (Nöllenburg Master'sThesis'05)

MetroMapLayout *is NP-hard.*

Proof.

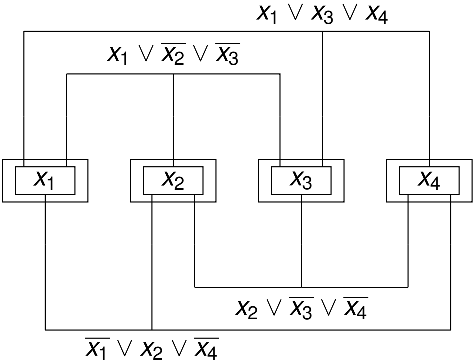
Reduction from Planar 3-Sat to MetroMapLayout. S □

Outline of the Reduction



Input: planar 3-SAT formula $\varphi = (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \ldots$

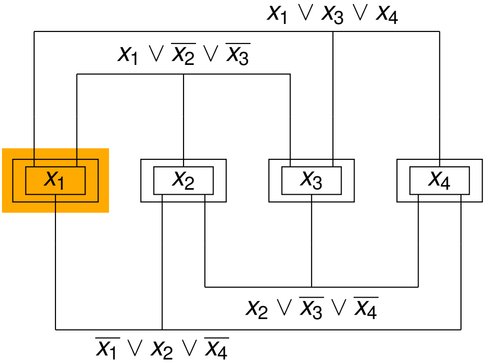
Outline of the Reduction



Input: planar 3-SAT formula $\varphi =$
$(x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \ldots$

Goal: planar embedded graph G_φ with:
G_φ has a metro map drawing $\Leftrightarrow$ φ satisfiable.

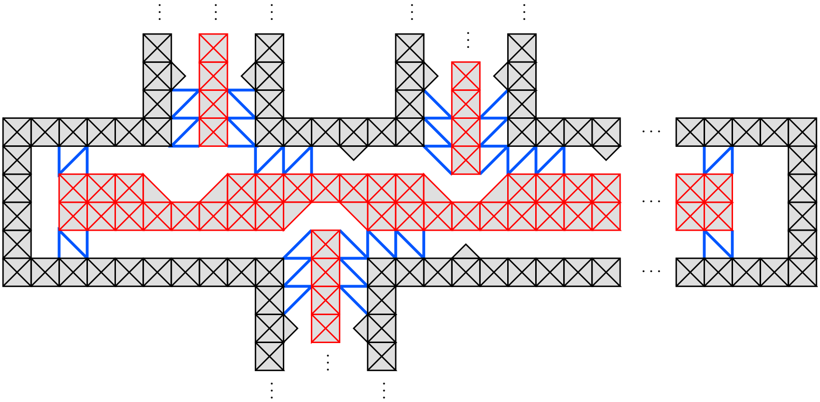
Outline of the Reduction



Input: planar 3-SAT formula $\varphi =$
$(x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \ldots$

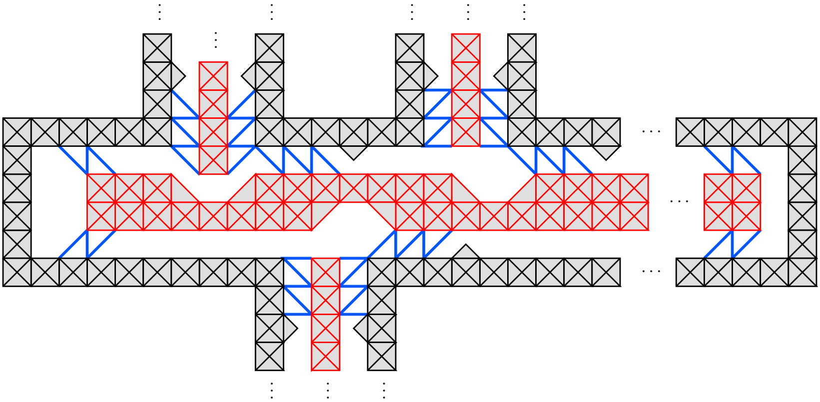
Goal: planar embedded graph G_φ with:
G_φ has a metro map drawing $\Leftrightarrow$ φ satisfiable.

Variable Gadget



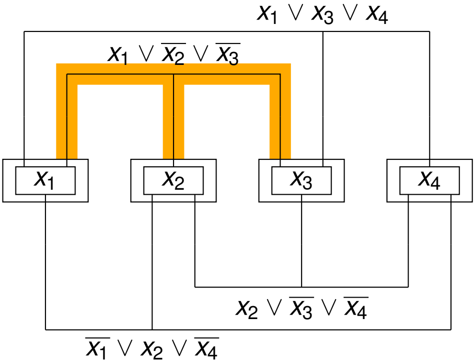
$x = \text{true}$

Variable Gadget



x = false

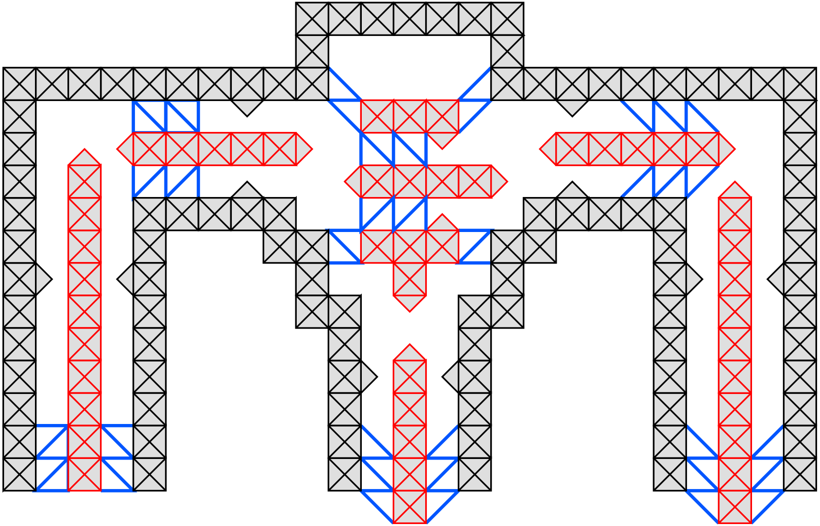
Outline of the Reduction



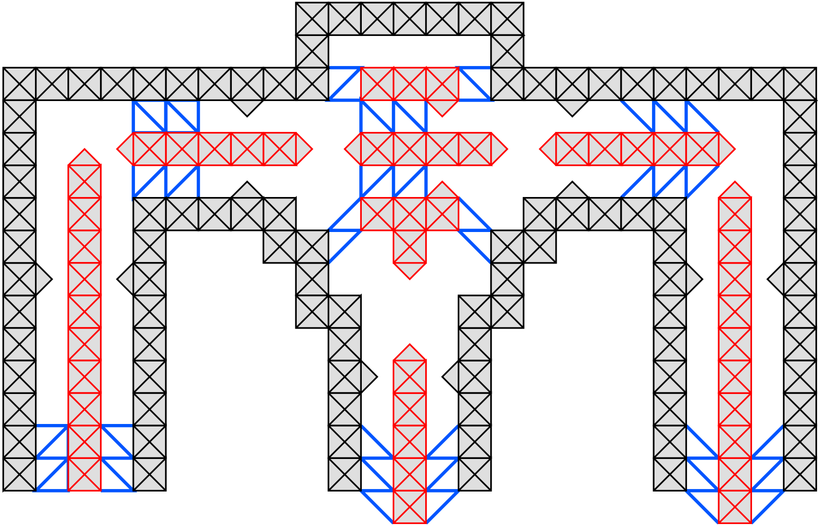
Input: planar 3-SAT formula $\varphi =$
$(x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \ldots$

Goal: planar embedded graph G_φ with:
G_φ has a metro map drawing $\Leftrightarrow$ φ satisfiable.

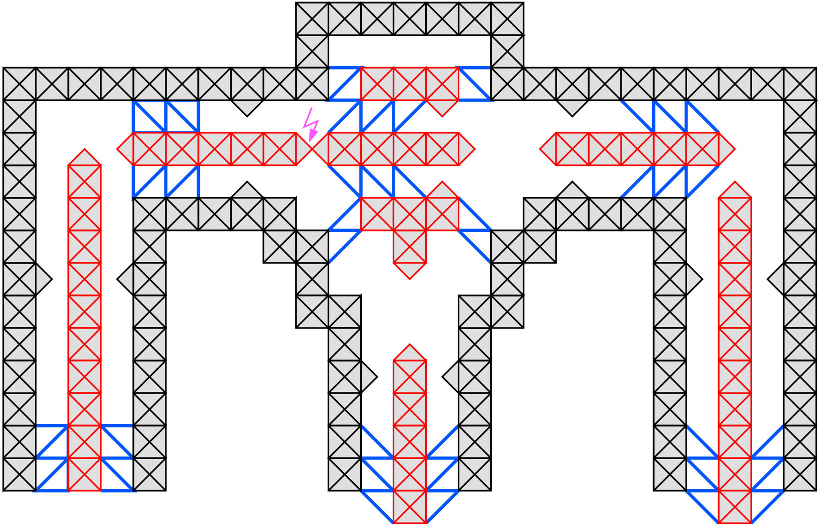
Clause Gadget



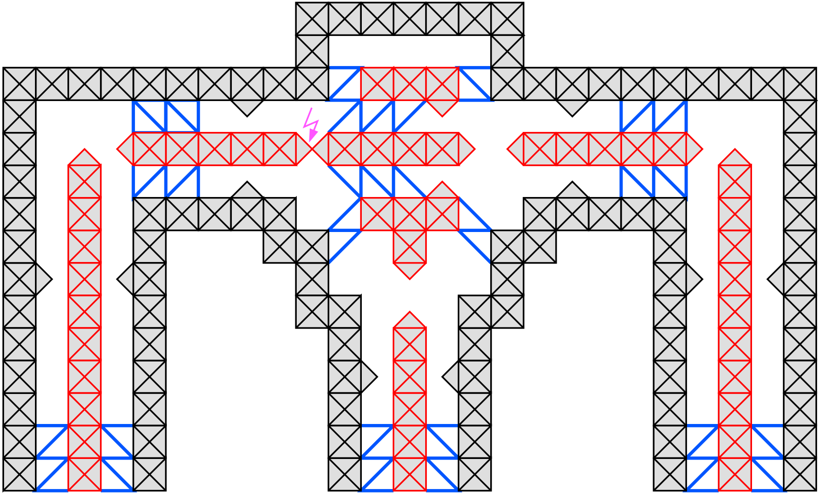
Clause Gadget



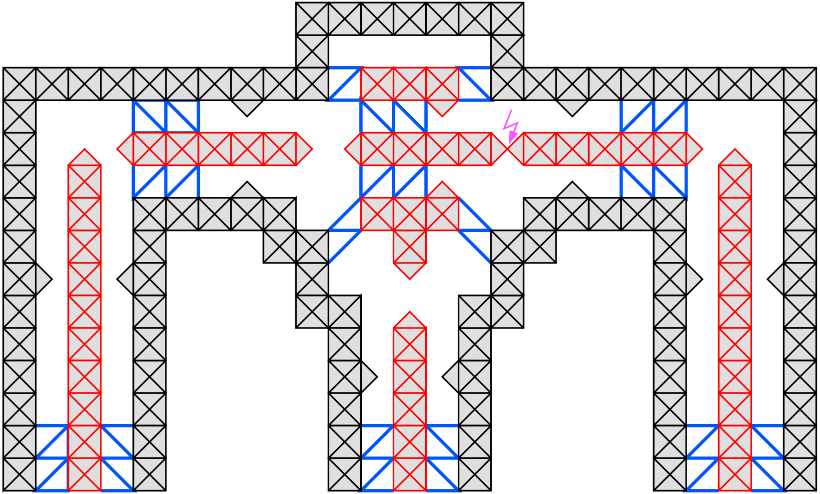
Clause Gadget



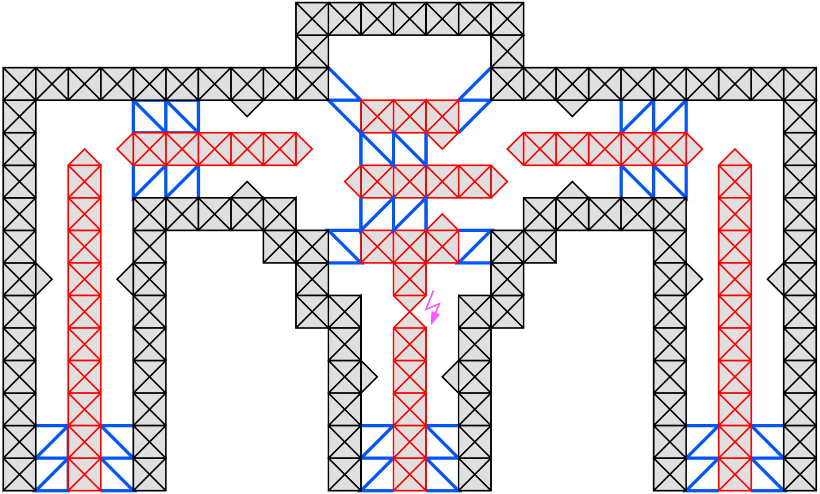
Clause Gadget



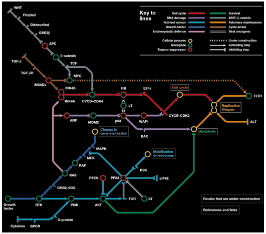
Clause Gadget

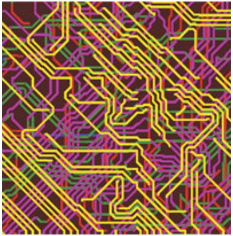


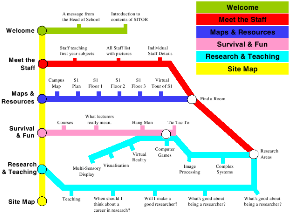
Clause Gadget

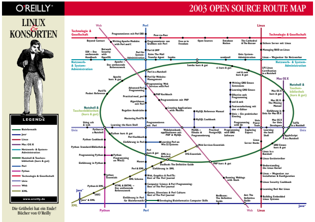


Other applications

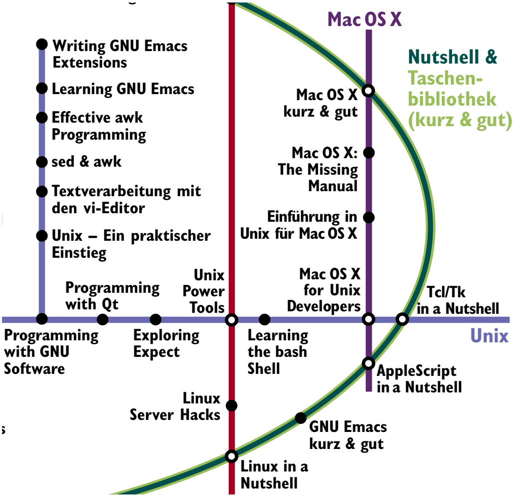




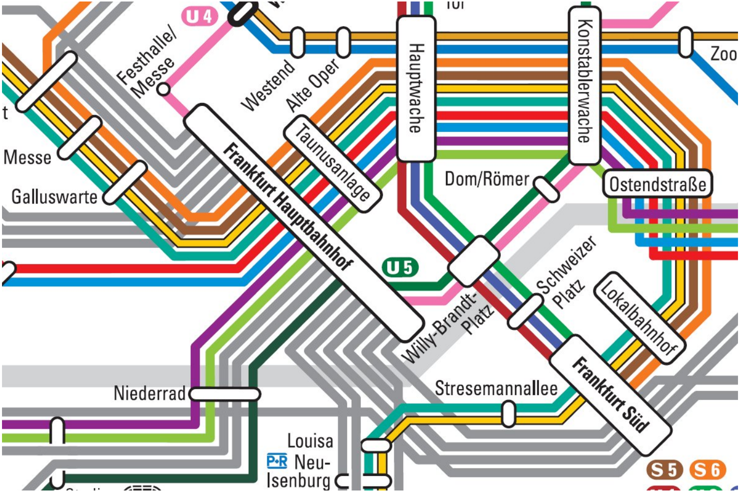




Clipping


Mac OS X
Nutshell &
Taschen-
bibliothek
(kurz & gut)
Writing GNU Emacs Extensions
Learning GNU Emacs
Effective awk Programming
sed & awk
Textverarbeitung mit den vi-Editor
Unix – Ein praktischer Einstieg
Mac OS X kurz & gut
Mac OS X: The Missing Manual
Einführung in Unix für Mac OS X
Mac OS X for Unix Developers
Programming with Qt
Unix Power Tools
Tcl/Tk in a Nutshell
Unix
Programming with GNU Software
Exploring Expect
Learning the bash Shell
AppleScript in a Nutshell
Linux Server Hacks
GNU Emacs kurz & gut
Linux in a Nutshell

To do: rectangular stations & multi-edges



Summary (metro maps)

- METROMAPLAYOUT is NP-hard.
- Formulated and implemented MIP.
- Our MIP can draw *any* kind of sketch “nicely”.
- Results comparable to manually designed maps.
- Reduced MIP size & runtime drastically.

Summary (metro maps)

- METROMAPLAYOUT is NP-hard.
- Formulated and implemented MIP.
- Our MIP can draw *any* kind of sketch “nicely”.
- Results comparable to manually designed maps.
- Reduced MIP size & runtime drastically.

To Do

- rectangular stations
- multi-edges
- user interaction (e.g., fixing certain edge directions)